ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA (ICAI)
GRADO EN INGENIERÍA TELEMÁTICA

Development of a Mobile Satellite Ground Communications Station

Autor: Javier Agüera Reneses
Director: Christopher Kitts

Madrid
Junio 2014

ACKNOWLEDGEMENTS

First and foremost, I would like to thank my team members Paulo Borges, Andrew Clavijo, Michael Kunis, Alex Mulcahy and Kristopher Sanford. Their inestimable contributions are integral part of this work.

In addition, I would like to deeply thank Professor Christopher Kitts for his guidance and help in developing, designing and accomplishing this project, lab space to work in, grant money to accomplish the goals and mentorship throughout the process.

I would like to extend my gratitude and the one from the rest of my team members to:

- Mike Rasay, for the assistance, advice, inspiration and mentorship that he graciously supplied during the course of the project.
- Thomas Adamek for the numerous times he assisted the project in multiple capacities as well as his generosity in lending hardware, parts and work space.
- The Undergraduate Engineering Programs department for their generous grant which funded the entire construction of our system.
- The Robotic Systems Laboratory for the facilities and assistance throughout the process.
- Professor Timothy Hight, for the assistance and guidance of throughout the design process, both inside and outside of class.
- Don MacCubbin for his assistance in using the machine shop, his guidance in ensuring we would properly using the machines and his insight into improving and manufacturing our designs.
- Anthony Young for their assistance in setting up and using code they had previously written which enabled communication to our rotors and orbit propagation.
- John Malsbury from Ettus Research, for his voluntary assistance and invaluable advice which was key for the development of the Software-Defined Radio subsystem.
- The many other members of the Robotic Systems Laboratory who helped us as we navigated our way through the design of the mobile ground station.

Finally, my deepest gratitude to the many family members, friends and faculty members who supported the team throughout the several months employed for the development of the project.

DEVELOPMENT OF A MOBILE SATELLITE GROUND COMMUNICATIONS STATION

Author: Agüera Reneses, Javier.
Supervisor: Kitts, Christopher.
Collaborating Entities: Santa Clara University Robotic Systems Lab; NASA Ames Research

ABSTRACT

The current ground control architecture at Santa Clara University uses static facilities to communicate with nanosatellites, confining the satellite contacts to whenever the satellite is overhead. In order to assist operations, a mobile satellite communication station was developed to increase the amount of contact time with the satellite. The station has S-band communication capabilities as well as APRS amateur band reception capabilities handled by a software-defined radio. To increase mobility, a stowage mechanism was designed to speed up setup and breakdown processes and an antenna auto-calibration system was created to eliminate the dish calibration process. The station is able to send commands to the satellite as well as receive mission data.

Keywords: SDR, satellite, nanosatellite, S-band, groundstation, APRS

1. Introduction

The appearance of nanosatellites together with advances in the field of space technologies during the last decade has enabled smaller private companies and research institutions to enter the spatial ecosystem at a fraction of the cost compared to previous times. Santa Clara University serves as an important research and operations hub through its Robotics Systems Labs and its partnership with the NASA Ames Research Park and other educational institutions. However, its fixed ground communications facilities are recently experiencing limitations because of the amount of simultaneous missions and experiments to track, and the needs to accommodate different satellital orbits. As a solution to this issue, a team of 6 undergraduate student researchers was assembled in order to create a first-of-its-kind mobile station by retrofitting an RV trailer. Simultaneously, some of the latest technologies such as Software-Defined Radio were introduced to substitute older system integrations, not only solving the original problem but also serving as a proof-of-concept for future designs.

2. Work definition

The final product requested by NASA and Santa Clara University (SCU) Robotics Systems Lab (RSL) consisted on an RV trailer fitted with equipment capable of fully replicating the capabilities of the fixed ground stations operated by the university in its California and El Salvador locations. It should be operated by a maximum of 2 people, and the RV should be

able to host them over a week with sleeping, cooking and personal hygiene facilities. The whole vehicle is powered by diesel generators, so reducing power consumption to the minimum feasible was a clear aim throughout the entire design and implementation process.

There are two main communication technologies used with nanosatellites. First is S-band communications, which is a bidirectional link employed to upload commands and download mission and experiment data. The S-band subsystem requires a relatively large dish antenna to operate. The other subsystem is the beacon receiver. Beacons are transmitted using the 437 MHz amateur radio band with the APRS/AX.25 format and contain information about telemetry, "health" of the aircraft, and in some occasions experiment data. It normally uses a 7-element Yagi antenna to operate.

The system must be able to track the satellites in real time, and provide accurate dynamic pointing of the antennas towards the position of the satellite accurately in real time. While the beacon system has relatively high pointing tolerance, the S-band communications do require a minimal margin of error of just a few degrees. In addition, since the vehicle might be parket in any direction and with variable inclination, one of the key features of the system is to be able to identify the position by GPS, the pointing direction of the vehicle by magnetometers, and the pitch of the antenna support by accelerometers.

One of the requisites was to introduce a new technology called Software-Defined Radio (SDR) as a proof of concept, substituting the old beacon receiving subsystem (OSCAR). SDR is the latest technology is radio communications. In short, it is a piece of hardware consisting on programmable logic that together with the help of a PC-grade microprocessor can be turned into any kind of radio appliance (cellphone base station, RADAR, digital TV transmitter, etc.). This allows extreme upgradeability without the need to replace entire pieces of expensive dedicated equipment, and cutting costs of the system by half.

3. Description of the product design

The mobile nanosatellite operations station uses the same architecture as the Robotics System Laboratory (RSL) for data, tracking, and interfacing but is located within a towable trailer. The main difference between both stations is that our design has been optimized for mobility, and every element on board has been modularized. Technologically the main differentiator is the use of SDR for the beacon system substituting the old HAM Radio based architecture. In the future, the SDR system will be upgraded to substitute other parts of the system such as the S-band transceiver. The S-band dish has also been reduced in size from the RSL's 3 meter mesh dish antenna to a 2.4 meter solid surface aluminum dish antenna to ensure a mobile solution could be realized. Finally, the entire satellite tracking system had to be redesigned in order to account for the new system dynamic conditions such as variable coordinates, inclination and orientation. These three subsystems (SDR, S-band dish and rotor auto-calibration) constituted the core of the development of the project.

My work was specialized on the development of the SDR system. For this purpose, we used a state-of-the-art USRP (Universal Software Radio Peripherial) from Ettus Research, linked to a standard Linux PC via USB 3.0. It was paired through one of its interfaces to a preamplifier and a 7-element Yagi antenna. This whole setup had a total cost of less than $1,000, which was half of the former legacy system. The core piece of software developed (besides the hardware installation) was the SCU BeaconStation. It consisted of a GUI Control Panel, with full digital signal processing underneath, as well as demodulation and packet decoding for AX.25 data format. The system was capable of outputting decoded beacon packets in plain text, uploading them to a server in real time, or transferring them to Data Turbine, which is used by NASA and RSL to exchange mission control data.

One of the star features of the system is its Doppler effect correction mechanism. Much like the change of sound pitch experienced by an observer when an ambulance passes by, there is a notable shift of frequency (up to 20KHz) when the satellite passes overhead. This makes impossible to track a signal unless real time correction is applied. Originally the system implemented a PLL frequency peak tracking system, but it was fully replaced because of poor results by a newly developed software named SCU PredictLink that generated a real-time mathematical prediction every 50 milliseconds based on TLE Keplerian orbital data of the satellite provided by NASA.

As an additional feature, an iPhone application was developed to be able to access beacon data in real time and control the system output remotely.

Results

A working design was presented and demonstrated during the 44th Annual Senior Design Conference at SCU, and it is expected that the finalized station will be operational in real missions during Fall 2014 after a series of end-to-end testing phases. The budget for the project was $21,500. The entire cost of the system was $20,375.

The mobile ground station has been found to increase contact availability by over 70% just by operating as another static facility at the same latitude as the current RSL station. With the mobility advantage, availability has the potential to dramatically increase. Increasing contact availability helps ensure minimum success criteria is met for the nanosatellite missions and allows for more complicated missions to be run in the future.

The software-defined radio presents a novel approach to beacon reception. It allows for simultaneous multiple satellite accommodation which will help the RSL cope with the increasing number of satellite missions it is being contracted. This system can be automated to track satellites. In addition, it offers a more compact, power-saving and upgradeable solution than a traditional dedicated system.

While the major design goals were attained, the system was not fully tested and debugged at the time of finalizing the document. The SDR subsystem managed to get a successful contact under real life conditions, but it was tested with a larger Yagi antenna. Introducing the more compact 7-element Yagi that is build on-board will require additional fine-tuning of the system. Finally, the mounting and stowage system for the dish antenna was finished just in time, and is likely to go through further revisions as live driving safety and full assembly tests are performed.

4. Conclusions

The successful development of this project will enable Santa Clara University and its partner institutions to increase their outreach, being able to grow in number of simultaneous missions and experiments, and increasing their operational efficiency. In addition, by renting the mobile station to private corporations and other educational institutions, its potential positive impact on society can be maximized.

The reasonably successful proof-of-concepts of new approaches with cutting-edge technologies such as SDR will enable dramatic cost reduction in the future as well as drive upgradeability and flexibility of the platform overall.

5. References

1. C. Kitts, M. Rasay, I. Mas, P. Mahacek, G. Minelli, J. Shepard, and J. Acain, "Respon- sive Small Satellite Mission Operations Using An Enterprise-Class Internet-Based Command and Control Network." in Proceedings AIAA Space Conference & Exposi- tion, pp. 1-10, San Diego, CA, 2008.

2. D. M. Jansky, M. C. Jeruchim, Communication Satellites in the Geostationary Orbit, 1st ed. Dedham, MA: Artech House, 1983.

3. E. Rodrigues, A. Rocha, D. Adrian, "Using GNURadio for a Satellite Beacon Detec- tor" in Proc. 15th Ka and Broadband Communications Navigation and Earth Obser- vation Conf., pp. 152-160, Cagliary, ITA, 2009.

4. U. Kuhar, G. Kandus, A. Vilhar, "Lowcost frequency stable beacon receiver based on software defined radio," Software, Telecommunications and Computer Networks (SoftCOM), 2012 20th International Conf., pp.1,5, Split, Croatia, 2012.

DESARROLLO DE UNA ESTACIÓN MÓVIL DE COMUNICACIONES SATELITALES

Autor: Agüera Reneses, Javier.
Director: Kitts, Christopher.
Entidades Colaboradoras: Santa Clara University Robotic Systems Lab; NASA Ames Research

RESUMEN DEL PROYECTO

La arquitectura actual de la estación terrestre de comunicaciones satelitales (groundstation) empleada por Santa Clara University emplea instalaciones estáticas para establecer comunicaciones con nanosatélites, limitando dichas comunicaciones a las ocasiones en las que el satélite atraviesa el cielo a la vista. Con el fin de asistir en las operaciones, se ha desarrollado una estación móvil de comunicaciones satelitales con el fin de aumentar el tiempo de contacto con el satélite. La estación tiene equipamiento de comunicación en banda S, así como capacidades de recepción de paquetes APRS en banda amateur llevada a cabo por tecnología SDR (Radio Definida por Software). Para aumentar su movilidad se diseñó un mecanismo de despliegue para acelerar el montaje y guardado del mismo, y un mecanismo de auto-calibración del rotor de las antenas. La estación es capaz de enviar comandos a un satélite así como recibir datos de la misión.

Palabras clave: SDR, satélite, nanosatélite, banda S, groundstation, APRS

1. Introducción

La aparición de nanosatélites junto a los avances en el campo de las tecnologías espaciales durante la pasada década han permitido que pequeñas empresas privadas y centros de investigación puedan entrar en el ámbito espacial con una fracción de los costes si se hace la comparación con épocas anteriores. Santa Clara University es un importante centro de operaciones e investigación a través de su Robotic Systems Lab y su alianza con el NASA Ames Research Park y otras instituciones académicas. Sin embargo, su infraestructura fija de comunicaciones satelitales está sufriendo recientemente limitaciones debido al elevado número de misiones simultáneas y experimentos a los que dar seguimiento, así como la necesidad de acomodar diferentes órbitas al mismo tiempo. Como solución a este problema, se formó un equipo de 6 estudiantes investigadores para crear una estación móvil pionera empleando un tráiler-caravana. Al mismo tiempo se introdujeron algunas de las últimas tecnologías como SDR para sustituir equipamiento y configuraciones obsoletos, no solo solucionando el problema original pero también sirviendo de prueba de concepto para diseños futuros.

2. Definición del trabajo

El desarrollo solicitado por la NASA y el Robotic Systems Lab (RSL) de Santa Clara University (SCU) consiste en un tráiler-caravana provisto de equipamiento capaz de replicar al máximo las capacidades de las estaciones terrestres fijas operadas por la universidad en sus localizaciones en California y El Salvador. Debe ser operado por un máximo de 2 personas, y el vehículo debe ser capaz de albergar por más de una semana a sus habitantes, con instalaciones de cocina y aseo. Todo el equipamiento es alimentado por generadores diesel, por lo que reducir el consumo de energía al mínimo factible fue un objetivo claro a lo largo de todo el proceso de diseño e implementación.

Hay dos tecnologías de comunicación principales que se emplean con nanosatélites. En primer lugar se encuentran las comunicaciones de banda S, consistentes en un enlace bidireccional empleado para cargar comandos y descargar datos experimentales o de la misión. El subsistema de banda S requiere una antena parabólica de tamaño relativamente grande para operar. El otro subsistema es el receptor de radiobalizas. Las radiobalizas se transmiten usando la banda de radioaficionados 437 MHz con el formato APRS/AX.25 y contienen información acerca de la telemetría, la "salud" de la aeronave, y en algunas ocasiones los datos del experimento. Normalmente se utiliza una antena Yagi de 7 elementos para su funcionamiento.

El sistema debe ser capaz de rastrear los satélites en tiempo real, y proporcionar direccionamiento preciso de las antenas hacia la posición del satélite en tiempo real. Mientras que el sistema de radiobalizas tiene tolerancia relativamente alta en este aspecto, las comunicaciones de banda S sí requieren un mínimo margen de error de sólo unos pocos grados. Además, ya que el vehículo puede ser aparcado en cualquier dirección y con inclinación variable, una de las características fundamentales del sistema es ser capaz de identificar la posición por GPS, la dirección en que apunta el vehículo empleando magnetómetros y la inclinación de la soporte de la antena con acelerómetros.

Uno de los requisitos era introducir una nueva tecnología llamada Radio Definida por Software (SDR) como una prueba de concepto, sustituyendo el antiguo subsistema de radiobalizas (OSCAR). SDR es la última tecnología en comunicaciones de radio. Se trata de un hardware compuesto por lógica programable que, junto con la ayuda de un microprocesador de PC, puede convertirse en cualquier tipo de aparato de radio (estación base de telefonía móvil, RADAR, transmisor de TDT, etc.) Esto permite actualizar el sistema en cualquier momento sin apenas costes añadidos ni la necesidad de desechar equipamiento obsoleto, así como reducir el precio del sistema a la mitad.

3. Descripción del modelo/sistema/herramienta

La estación móvile de counicaciones satelitales utiliza la misma arquitectura que el Robotic Systems Lab (RSL) para la transmisión de datos, el seguimiento de datos y la interconexión, pero se encuentra dentro de un vehículo. La principal diferencia entre ambas estaciones es que nuestro diseño se ha optimizado para la movilidad, y todos los elementos a bordo se han modularizado. Tecnológicamente el factor diferencial principal es el uso de SDR para el sistema de radiobalizas sustituyendo la vieja arquitectura basada en HAM Radio. En el futuro, el sistema SDR se actualizará para sustituir otras partes del sistema como el

transceptor de banda S. En el subsistema de banda S también se ha reducido en tamaño de la antena parabólica de malla de 3 metros del RSL, empleándose finalmente una antena parabólica de aluminio con superficie sólida y 2,4 metros de diámetro con el fin de lograr una movilidad óptima. Por último, todo el sistema de seguimiento de satélites tuvo que ser rediseñado con el fin de tener en consideración las nuevas condiciones dinámicas del sistema como coordenadas variables, inclinación y orientación. Estos tres subsistemas (SDR, antena de banda S y auto-calibración del rotor) constituyeron el núcleo de tareas fundamentales en el desarrollo del proyecto.

Mi trabajo se especializó en el desarrollo del sistema SDR. Para ello, se utilizó la última generación de USRP (Periférico Universal de Radio por Software) desarrollado por Ettus Research, vinculado a un PC Linux estándar a través de USB 3.0. Una de sus interfaces se conectó a un preamplificador y finalmente a una antena Yagi de 7 elementos. Toda esta instalación tuvo un coste total de menos de 1.000 dólares americanos, siendo ésta la mitad del anterior sistema. La pieza central del software desarrollado (además de la instalación de hardware) fue *SCU BeaconStation*. Se compone de un panel de control GUI, con procesamiento digital completo de la señal así como demodulación y decodificación de paquetes en formato de datos AX.25. El sistema es capaz de dar salida a los paquetes de radiobaliza decodificados en texto plano, subirlos a un servidor en tiempo real, o transmitirlos a Data Turbine, un software que se utiliza por la NASA y RSL para el intercambio de datos de control de misiones.

Una de las características estrella del subsistema es su mecanismo de corrección del efecto Doppler. Al igual que la variación en el tono del sonido experimentado por un observador cuando una ambulancia pasa por delante suyo, hay un cambio notable de frecuencia (hasta 20 kHz) cuando el satélite pasa por encima de la estación. Esto hace imposible el seguimiento de una señal si no se aplica una corrección en tiempo real. Originalmente, el sistema implementado un sistema de seguimiento de picos de frecuencia por PLL, pero fue reemplazado tras los malos resultados en las pruebas por un nuevo desarrollo de software llamado *SCU PredictLink* que genera una predicción matemática en tiempo real cada 50 milisegundos basándose el los datos orbitales Keplerianos TLE del satélite proporcionados por la NASA.

Como característica adicional, se desarrolló una aplicación de iPhone para ser capaz de acceder a los datos de radiobaliza en tiempo real y monitorizar el sistema.

4. Resultados

Se presentó y demostró una versión funcional del producto durante la 44ª Conferencia Anual de Proyectos Finales en SCU, y se espera que la estación finalizada esté operando en misiones reales durante el otoño 2014 una vez finalizada una serie de fases de integrales. El

presupuesto para el proyecto fue de 21.500 dólares mientras que su costo total fue de 20.375 dólares.

En pruebas empíricas se ha logrado demostrar un incremento de más de 70% empleando la estación móvil como otra instalación estática en la misma latitud que la estación de RSL actual. Si se tuviese en cuenta el factor de movilidad, existe el potencial de incrementar notablemente esta cifra. El aumento de la disponibilidad para contactos ayuda a garantizar el cumplimiento de unos criterios mínimos de éxito en misiones de nanosatélites, y permitirá ejecutar misiones más complejas en el futuro.

La tecnología SDR presenta un novedoso enfoque para la recepción de radiobalizas. Permite acomodar el seguimiento simultáneo de múltiples satélites, hecho que ayudará al RSL a hacer frente al creciente número de misiones para las cuales se están contratando sus servicios. Este sistema puede ser automatizado para el seguimiento de múltiples satélites y además ofrece un ahorro de energía siendo más compacto y actualizable que un sistema dedicado tradicional.

Si bien se han alcanzado los principales objetivos de diseño, las pruebas extensivas sobre el sistema no habían sido finalizadas pro ejemplo en la fecha de finalización de este documento. El subsistema de SDR consiguió un contacto exitoso en condiciones reales, con la excepción de que dicha prueba se ejecutó con una antena Yagi más grande. El empleo de la antena Yagi de 7 elementos más compacta que se encontrará a bordo requerirá ajustes adicionales del sistema. Por último, el montaje y el sistema de almacenamiento para la antena parabólica se terminó con poco margen de tiempo, y es probable que deba pasar por nuevas revisiones como resultado de las pruebas de seguridad al conducir el vehículo y de montaje completo.’

5. Conclusiones

La implementación exitosa de este proyecto permitirá a la Universidad de Santa Clara y sus instituciones asociadas aumentar su capacidad en el ámbito espacial, siendo capaz de crecer en número de misiones y experimentos simultáneos así como aumentando su eficiencia operativa. Además, mediante el alquiler de la estación móvil a empresas privadas y otras instituciones educativas se peude maximizar su potencial de impacto positivo en la sociedad.

El éxito razonable de la prueba de concepto de nuevos enfoques basados en tecnologías de última generación como SDR permitirán la reducción dramática de costes en el futuro, así como sentar las bases para la actualización futura de la plataforma entera de hardware y aumentar su flexibilidad.

6. Referencias

1. C. Kitts, M. Rasay, I. Mas, P. Mahacek, G. Minelli, J. Shepard, and J. Acain, “Responsive Small Satellite Mission Operations Using An Enterprise-Class Internet-Based Command and Control Network.” in Proceedings AIAA Space Conference & Exposi- tion, pp. 1-10, San Diego, CA, 2008.

2. D. M. Jansky, M. C. Jeruchim, Communication Satellites in the Geostationary Orbit, 1st ed. Dedham, MA: Artech House, 1983.

3. E. Rodrigues, A. Rocha, D. Adrian, “Using GNURadio for a Satellite Beacon Detec- tor” in Proc. 15th Ka and Broadband Communications Navigation and Earth Obser- vation Conf., pp. 152-160, Cagliary, ITA, 2009.

4. U. Kuhar, G. Kandus, A. Vilhar, “Lowcost frequency stable beacon receiver based on software defined radio,” Software, Telecommunications and Computer Networks (SoftCOM), 2012 20th International Conf., pp.1,5, Split, Croatia, 2012.

Table of Contents

1 Introduction 1
- 1.1 Background . . . 1
- 1.2 RSL Operations . . . 2

2 State of the Art 5

3 Project Statement and Scope 9

4 Systems-Level Design 11
- 4.1 Overview . . . 11
- 4.2 Customer Needs and System Requirements . . . 16
- 4.3 Refined Project Concept . . . 18
- 4.4 Benchmarking Results . . . 19
- 4.5 Functional Analysis . . . 20
  - 4.5.1 *S-Band Parabolic Dish Antenna* . . . 21
  - 4.5.2 *Software Defined Radio* . . . 22
  - 4.5.3 *Auto-Calibration* . . . 22
- 4.6 Key System Level Issues and Tradeoffs . . . 23
  - 4.6.1 *Size Reduction* . . . 23
  - 4.6.2 *Dish Case* . . . 24
  - 4.6.3 *Panel Assembly* . . . 24
  - 4.6.4 *Flower Collapsing Dish* . . . 24
  - 4.6.5 *Hinge Collapsing* . . . 25
  - 4.6.6 *Telescoping-Hinged Arm* . . . 25
- 4.7 Team and Project Management . . . 25
  - 4.7.1 *Project Challenges* . . . 25
  - 4.7.2 *Budget* . . . 26
  - 4.7.3 *Design Process* . . . 26
  - 4.7.4 *Risks and Mitigations* . . . 26
  - 4.7.5 *Team Management* . . . 27

5 S-Band Antenna 28
- 5.1 Overview . . . 28
- 5.2 Requirements . . . 28
- 5.3 Options and Tradeoffs . . . 29
- 5.4 Detailed Design Description . . . 30
  - 5.4.1 *Component Breakdown* . . . 30
- 5.5 Supporting Analysis . . . 45
- 5.6 System Progress and Future Testing . . . 47

6 Software-Defined Radio 48
- 6.1 Overview . . . 48

6.2 Requirements of the Software-Defined Radio . . . 52
6.3 Options and Tradeoffs Summary . . . 53
6.3.1 *FunCube Dongle* . . . 54
6.3.2 *USRP N200* . . . 55
6.3.3 *USRP B200* . . . 56
6.3.4 *Laptop Computer versus Micro Controller* . . . 57
6.3.5 *Doppler Correction Solution Tradeoffs* . . . 58
6.3.6 *Consideration of Post-Demodulating Filter Implementation* . . . 59
6.4 Detailed Design Description . . . 60
6.4.1 *Digital Signal Processing* . . . 60
6.4.2 *Signal Sampling and Recording* . . . 61
6.4.3 *Doppler Shift Correction* . . . 61
6.4.4 *Noise Reduction and Signal Conditioning* . . . 65
6.4.5 *Demodulation, Post-Filtering and Decoding* . . . 67
6.4.6 *Decoded Data Storage and Transmission* . . . 68
6.5 System Testing and Verification Procedures . . . 70
6.5.1 *Phase 1: SDR Equipment Pre-testing* . . . 71
6.5.2 *Phase 2: Beacon Reception Testing with Simulator* . . . 72
6.5.3 *Phase 3: Beacon Reception from O/OREOS Orbiting Satellite* . . . 73
6.5.4 *Current Development Status and Pending Work* . . . 74

7 Auto-Calibration 76
7.1 Overview . . . 76
7.2 System Options and Tradeoffs . . . 77
7.3 Design Description . . . 78
5.3.1 *Rotation Matrix* . . . 81
7.4 Supporting Analysis . . . 81
7.5 Test and Verification . . . 82

8 System Integration and Testing 86
8.1 Overview . . . 86
8.2 Workstation Integration . . . 88
8.3 Experimental Protocol . . . 89
8.4 Testing and Results . . . 90

9 Costing Analysis 93
9.1 Budget and Costs . . . 93

10 Business Plan 95
10.1 Executive Summary . . . 95
10.2 Background . . . 95
10.3 Key Technologies and Customers . . . 96
10.4 Marketing Strategies . . . 97
10.5 Production Cost and Price . . . 97
10.6 Service . . . 98

11 Engineering Standards and Realistic Constraints 99
11.1 Ethical Analysis . . . 99
11.2 Economics . . . 101
11.3 Manufacturability . . . 101
11.4 Legal . . . 102
11.5 Safety . . . 103
11.6 Usability . . . 104

12 Summary 105
12.1 Project Summary . . . 105

12.2 Future Development 106

References 109

A Product Design Specification 111
A.1 Questionnaires 111
A.2 Budget 113

B Block Diagrams 114

C Parts List and Design Drawings 116

D Specification Sheets 131

E Code 138
E.1 Auto-Calibration 138
E.2 Software-Defined Radio 141

List of Figures

1.1 RSL Ground Segment . . . 3
1.2 RSL Block Diagram . . . 4
1.3 GATR Mobile Communication System . . . 6
1.4 GlobeComm Global Storm Mobile Communication System . . . 7

2.1 Mission Architecture Diagram . . . 12
2.2 Three Meter Mesh Dish . . . 13
2.3 Trailer Cutaway View . . . 13
2.4 Mobile Ground Station Block Diagram . . . 15

3.1 S-band System Block Diagram . . . 31
3.2 S-band Antenna Hardware Assembly . . . 31
3.3 Assembled Mast Base Support Structure . . . 32
3.4 Exploded View of Mast Base Support Structure . . . 32
3.5 Assembled Support Structure on Trailer Tongue . . . 33
3.6 Antenna Mast Base Plate . . . 34
3.7 Dish Antenna Mast . . . 35
3.8 SPID RAS Rotor and Rotor Controller . . . 36
3.9 2.4 Meter Solid Surface Aluminum Dish Antenna . . . 37
3.10 Rotor to Dish Adapter . . . 37
3.11 Rotor to Dish Adapter Exploded View . . . 38
3.12 Dish Adapter Photo . . . 39
3.13 Dish Adapter Connected to Rotor Hub . . . 39
3.14 Feed Structure Assembly . . . 40
3.15 Feed Structure Exploded View . . . 41
3.16 Antenna Feed Structure Arms . . . 42
3.17 Feed Assembly to Dish Adapter . . . 42
3.18 Antenna Feed Structure Dish Attachment . . . 43
3.19 Dish Antenna Feed . . . 44
3.20 MicroHard 2420 Radio . . . 44
3.21 FEA Mast Analysis . . . 46
3.22 FEA Dish Analysis . . . 47

4.1 SDR Architecture . . . 49
4.2 Top SDR Filter Diagram . . . 50
4.3 Bottom SDR Filter Diagram . . . 51
4.4 Funcube Dongle Photo . . . 54
4.5 USRP N200 Photo . . . 55
4.6 USRP B200 Photo . . . 56
4.7 Raspberry Pi Photo . . . 57
4.8 Phase Locked Loop Diagram . . . 58
4.9 Predict Software Block Diagram . . . 59
4.10 DSP System Diagram . . . 60

4.11 SDR Filter Data Path . . . 62
4.12 SDR GUI Block Diagram . . . 65
4.13 DSP Filtering Block Diagram . . . 66
4.14 Quadrature Demodulator . . . 68
4.15 Example SporeSat Beacon Packet . . . 69
4.16 NOAA Image . . . 72

5.1 RSL vs Mobile Operations During Tracking . . . 76
5.2 Photo of APM 2.6 Sensor Package . . . 78
5.3 Auto-Calibration Block Diagram . . . 79
5.4 Physical Auto-Calibration Rotations . . . 80
5.5 Z-Y-Z Frame Transformation . . . 81
5.6 Auto-Calibration Testings Photos . . . 83

6.1 Complete Component Block Diagram of Mobile Communication Station . . . 87
6.2 SDR Block Diagram . . . 88
6.3 S-band Block Diagram . . . 89

B.1 Mobile Communication Station Block Diagram . . . 114
B.2 SDR Block Diagram . . . 115

C.1 Feed Assembly . . . 116
C.2 Adapter Assembly . . . 117
C.3 Support Structure Assembly . . . 118
C.4 Feed . . . 119
C.5 Feed Bracket . . . 120
C.6 Tripod First Arm . . . 121
C.7 Tripod Adapter . . . 122
C.8 Tripod Second Arm . . . 123
C.9 Dish Feed Attachment . . . 124
C.10 Adapter Base . . . 125
C.11 Adapter Wall . . . 126
C.12 Adapter Fin . . . 127
C.13 Support Frame . . . 128
C.14 Suport Link . . . 129
C.15 Support Frame Two . . . 130

D.1 APM 2.6 Specifications . . . 131
D.2 B200 Specification Sheet . . . 132
D.3 Dish Feed Frequency Specification . . . 133
D.4 MicroHard Radio Specification Sheet . . . 134
D.5 SPID RAS Specification Sheet . . . 135
D.6 Yaesu G-5500 Rotor Specification Sheet . . . 136
D.7 7-Element Yagi Antenna Specification Sheet . . . 137

List of Tables

3.1 Mast Properties 45
3.2 Dish Properties 46

4.1 SDR Tradeoffs Chart 56
4.2 SDR Requirements Chart 71

5.1 Azimuth and Elevation Corrected Values 82

7.1 Complete Cost Report for Mobile Communication Station 93

8.1 Price Points For Contracting Mobile Communication Station 97

A.1 Customer Needs Matrix 111
A.2 Complete Cost Report for Mobile Communication Station 113

Chapter 1

Introduction

1.1 Background

The design project is a mobile satellite communication station for the Robotic Systems Laboratory (RSL) at Santa Clara University to support ground control for nanosatellite missions. Supporting satellite missions is the main motivation for our project. Satellites provide many societal benefits including communication services, reconnaissance, weather analysis, earth observation, and support for scientific experiments. Our project was designed specifically to support nanosatellite missions. Nanosatellites are significantly smaller but have several advantages over larger scale satellites. They are cheaper, are quicker to manufacture and develop than larger satellites, are able to test new technologies that may not require a large scale satellite, and are much easier for groups in academia to be involved in their development (Milliano & Verhoeven, p. 1425). Small space programs also can benefit greatly from the use of satellites. Baker estimated that at least 80% of program goals can be achieved for 20% of the cost by using a nanosatellite (Baker, p. 301). These low costs have allowed 40 universities to engineer a nanosatellite within the past decade (Milliano & Verhoeven, p. 1425).

Since the 1990's, scientific researchers and engineers have been designing nanosatellite applications with considerable academic involvement. Missions such as TERRIERS, CATSAT, and SNOE were all early nanosatellite projects largely involved with the academic community (Baker, p. 302). The primary attraction of a nanosatellite over larger scale satellites is its cost. When considering the amount of potential academic research that could be done with the use of satellites it becomes clear why the push for nanosatellite

development is prominent (Milliano & Verhoeven, p. 1425). Due to the increased develop- ment of these nanosatellites, cheap ground operation solutions have also been developed to suit these missions. The continued nanosatellite development is due to successful ground control operation and realization of mission success criteria. Static stations such as the con- trol stations used by the RSL have been the ground segment solution, leveraging the S-band and amateur band frequencies to operate the nanosatellites. We believe that mobility in the ground segment for these nanosatellite missions is the next step toward communication development.

1.2 RSL Operations

The RSL is contracted by NASA nanosatellite programs to handle ground operations after launch. The operations team is responsible for downloading data, monitoring the health and status of the satellite, and sending commands to the satellite that are specified by the NASA engineers. The team has done the ground segment for several satellite missions including GeneSat-1, PharmaSat, and O/OREOS, which were all projects from the NASA Ames research center.

Currently the RSL operates from a static multi-antenna ground station location on the Santa Clara University campus. Given a nanosatellite in low earth orbit, the satellite will be visible in the sky several times a day for 1-15 minutes each time for any given station location.

During early mission operations the location of the satellite is difficult to determine and often times thwarts communication until high quality orbit information is provided to the operations team. An increasing number of satellite missions is also being contracted to RSL, which will put stress on the capabilities of a single station. Rather than creating another static station at a different location, a mobile ground station could be constructed with the same functionality as the current ground station. Using orbit propagation software and taking advantage of the satellite's orbit, the operations team would be able to move

the mobile ground station to a location that increases visibility time with the satellite. The mobility would increase the ability to find the spacecraft in early mission operation as well as increase the contact time with the satellite during the main mission phases.

The mission control ground segment can be seen in Figure 1.1. The RSL uses several antenna to allow it to communicate and receive data from the satellite. These antenna are controlled and monitored from central control consoles located at Santa Clara. The data is also stored locally and then sent to NASA engineers for processing. Several receive-only beacon stations are also located around the continental United States to allow for additional health and status monitoring of the satellite (Kitts, p. 4).

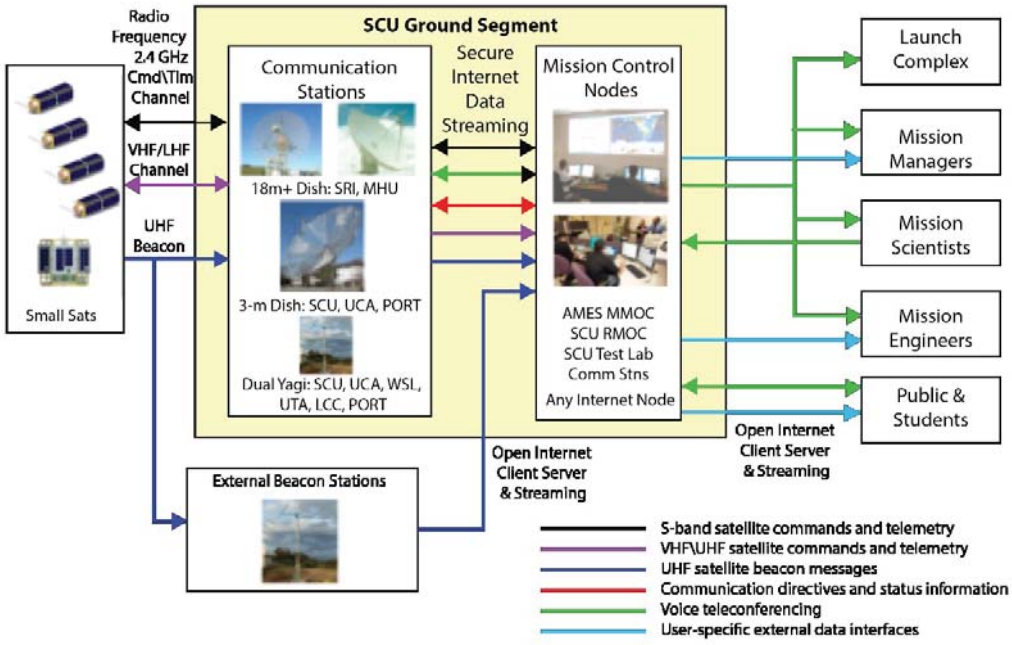


Figure 1.1: RSL nanosatellite ground segment architecture diagram (Kitts, p. 3).

The current ground station at SCU includes two S-band three-meter dishes, a UHF/VHF antenna, and a 7 element antenna. The S-band dishes are used to send commands and receive responses, which primarily contain mission data or the status of spacecraft subsystems. The UHF/VHF antenna and 7 element are used for beacon data that are sent out by the satellite. Several computers are used to control each dish as well as their tracking functionality. Two central computers host the databases and the operations

software. Figure 1.2 shows how all the communication components are connected to each other. The goal of the mobile communication station is to maintain the same architecture, functionality, and performance as the current ground station while adding the mobility component.

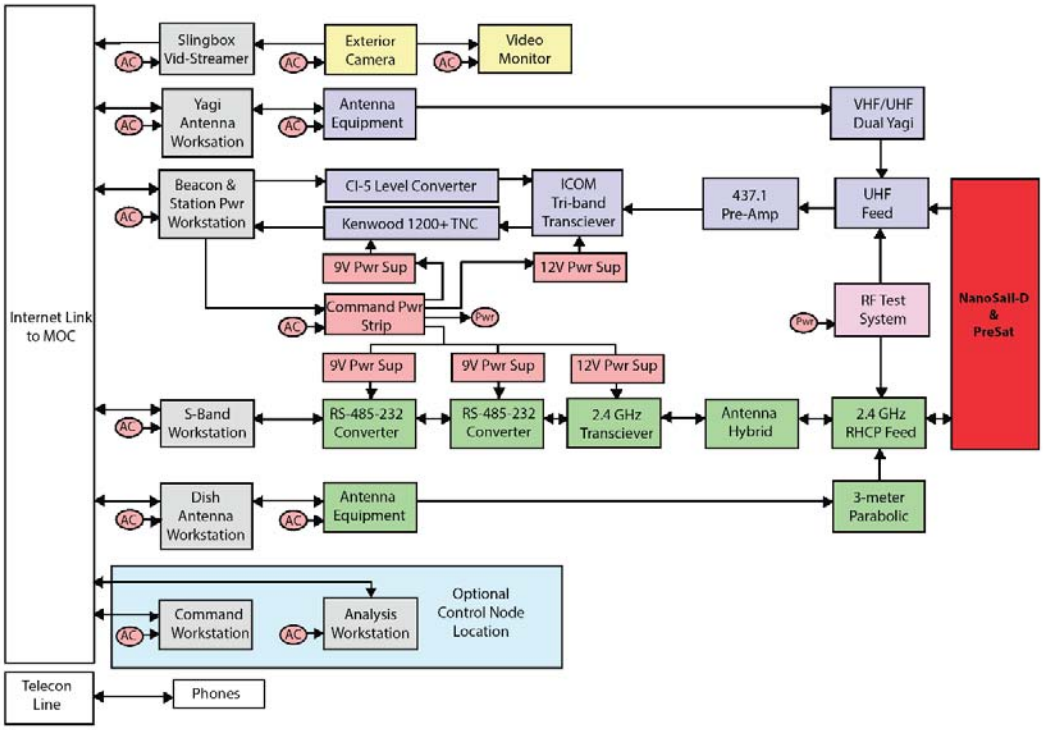


Figure 1.2: RSL Ground Station Block Diagram including S-band and UHF/VHF systems (Kitts, p.5).

Chapter 2

State of the Art

Mobile satellite communication is not a new innovation and has been developed for several applications. An intergovernmental organization under the name INMARSAT provided a worldwide maritime service through leasing a satellite system. Other possible uses include determining the position of airborne vehicles or communication in remote areas or areas that have been impacted by some disaster (Janksy, p. 22). Other projects have attempted to exploit the performance of a mobile communication station.

A van with an active communication channel to a satellite called MARECS tested the attenuation of sig- nal while the van was moving. As expected, the signal deteriorated due to buildings, trees, and other obstacles that were in the way creating a phenomena known as signal shadowing. Several copies of the signal are seen by the transceiver, effectively scrambling the signal. However, even with the signal attenuated from these factors, the study found that effective data transmission through a mobile satellite channel could be achieved (Lutz, p. 375). The signal shadowing effect will not be as much of a concern for our design because ideally the station will be parked in a location with the fewest surrounding obstacles.

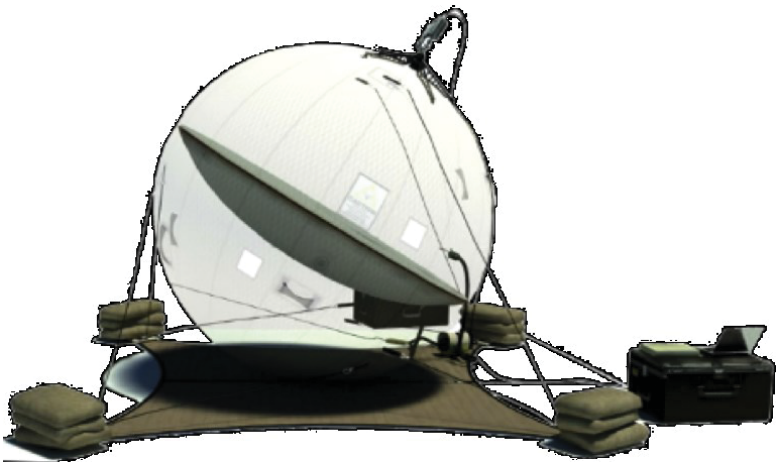


Figure 1.3: GATR's deployable antenna, the dish is held by a reflective inflatable sphere.

There are several systems similar to our design that currently exist, namely GATR's inflatable antenna and GlobeComm's GlobalStorm 2400. The GATR system is particularly innovative and uses an inflatable sphere to hold a 2.4 meter dish in the center as shown in Figure 1.3. Although the pointing mechanism is manual and the azimuth tracking can only be done +/- 10 from the orientation of original setup, it is considerably compact, fitting into two large suitcases when stowed, and very mobile.

The GlobeComm GlobalStorm 2400 seen in Figure 1.4 is considerably larger and supports 2.4 meter and 3.7 meter dishes. The entire system exists on a trailer where it is stowed and deployed from. Unlike the GATR system, it uses a rigid dish which folds in on itself when being stowed. The folding mechanism and the separate trailer housing allow for the dish to be rapidly deployed, on the order of minutes. It is a much heavier system as well and must be pulled by a vehicle. The size of this system is closer to our design.

Figure 1.4: GlobeComm Global Storm 2400 mobile satellite communication trailer.

Throughout its history of operating nanosatellites, the RSL has incorporated different solutions for its subsystem although the main architecture has for the most part remained constant (Kitts, pp. 3-6). The current active setup includes two S-band capable satellite dishes located at Santa Clara University. These dishes as well as the Microhard radios on them are all controlled via a central control station. A second type of antenna is located at Santa Clara University that handles UHF and VHF signals. This antenna is attached to a multi-band ICOM IC-910H transceiver that computes the signal's Doppler shift given the satellites Keplerian elements. The subsystem listens for beacon packets from the satellite and is also controlled from the central ground station. The RSL has had multiple mission successes using this system architecture, one example being GeneSat where the ground operations team was able to successfully fulfill primary mission criteria within a month of mission lifetime. When creating a mobile version of the RSL's ground station, we are willing to sacrifice some performance for mobility with the knowledge that the relocation of the system will allow us to make up for performance losses.

Chapter 3

Project Statement and Scope

The objective of this project was to build a mobile satellite communication station to assist the Robotics Systems Laboratory in their operations with nanosatellites. A mobile ground station increases the number of possible satellite contacts and helps assists in satellite identification in early orbit operations. A team composed of 6 multidisciplinary members was assembled. To create the mobile ground station, we equipped a towable trailer with an S-band communication and tracking station as well as with a receive-only beacon communication and tracking station.

The receive-only beacon system was implemented through a software defined radio in order to reduce the costs of the standard RSL beacon system solution. The S-band dish and support structure were de- signed for maximum mobility, allowing assembly and disassembly to take under a day for a crew of two people. An antenna auto-calibration system was implemented to remove the rotor calibration step and increase the overall mobility. The result of our progress this year is three separate subsystems that have undergone initial phases of testing. Although they have not been integrated with the mobile vehicle and tested for nominal operations, current functionality suggests that the current design will meet the needs of the RSL team. That team currently plans to fully integrate systems in order to have an operational system by Fall 2014.

An important aspect of the project is the validation of new technologies such as Software Defined Radio that will be the core of future infrastructure projects within the lab. A full analysis of upgradeability and expansion of the features which are currently part of

the SDR system was elaborated, and as a result all of the parts of the systems were designed with a modular approach.

During the course of the project I took active part in the design, development and testing of the various subparts of the projects, excepting those involving mechanical design and construction. In particular, I led the RF team which was in charge of the whole telecommunications part of the station. My area of specific and exclusive focus was the Beacon station using Software Defined Radio, which is described in Chapter 4, and constitutes one of the main breakthroughs and areas of future development of this project.

Chapter 4

Systems-Level Design

4.1 Overview

The mobile nanosatellite operations station uses the same architecture as the Robotics System Laboratory (RSL) for data, tracking, and interfacing but is located within a towable trailer. The mobile communication station differs from the current station used by the RSL because it is suited for mobility and uses a cheaper amateur band receiver solution in the form of a software defined radio. The S-band dish has also been reduced in size from the RSL's 3 meter mesh dish antenna to a 2.4 meter solid surface aluminum dish antenna to ensure a mobile solution could be realized. The final differing element of the mobile station is the auto-calibration of the rotors. The station is expected to operate while parked on various terrain. The auto-calibration system accounts for the unevenness of the terrain and factors it in to the dish tracking. This type of calibration is not necessary for the static station since the antenna remains fixed on level ground with a default pointing direction of North. The station will have the same role in the mission architecture as the current ground station as can be seen in Figure 2.1 while maintaining two methods of communication: S-band and amateur band. The station will operate independently of the other existing communication stations and control nodes but will use the same databases that the other control nodes use.

The S-band dish is the main method of communication with nanosatellites and is often the main contributor to mission success in the form of downloaded data. Performance was sacrificed for mobility by reducing the size of the dish from the RSL's 3-meter dish to

a 2.4 meter. The setup time of the 3 meter mesh dish, shown in figure 2.2 is on the order of a day due to a large number of pieces which is not ideal for a mobile station.

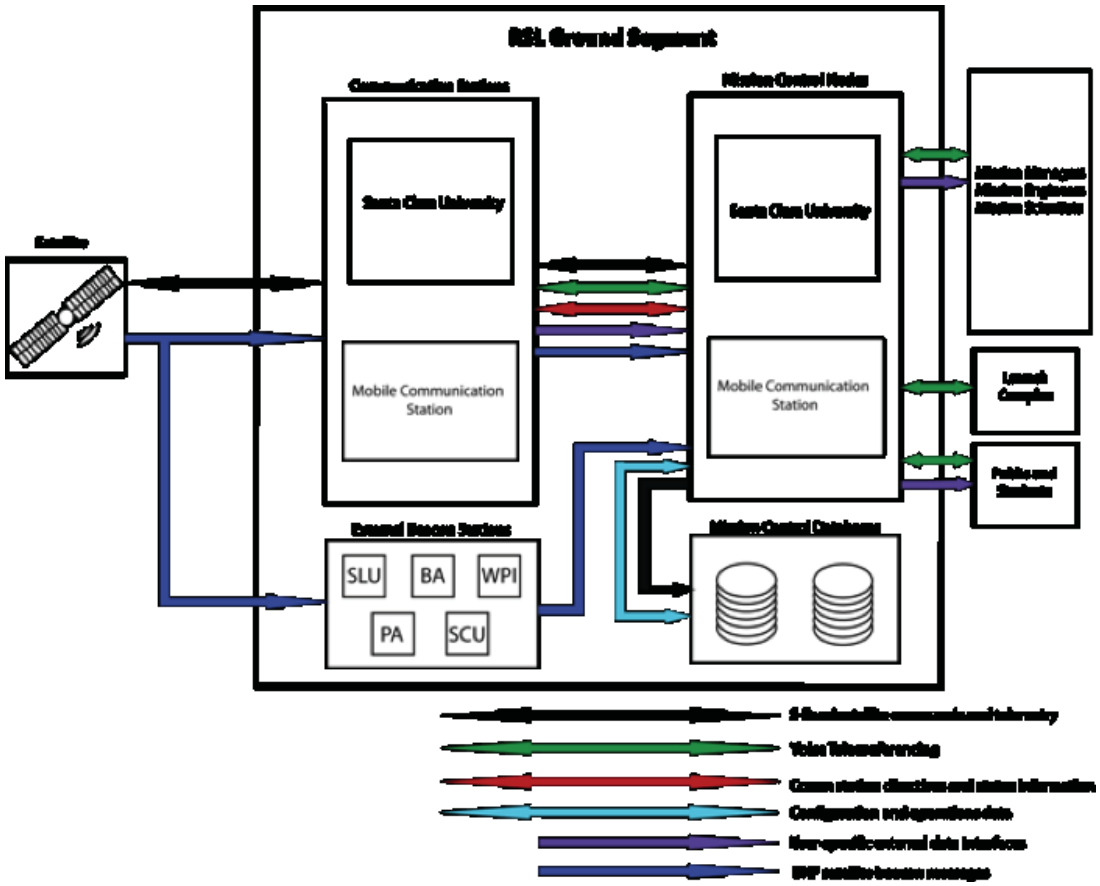


Figure 2.1: Mission Architecture Diagram for Mobile Satellite Communication Station

The 2.4 dish is assembled from four panels making it much more manageable for operations. Considerable development went into the interface between the trailer and the dish support structure. As seen in Figure 2.3 the dish is located on the tongue of the towable trailer leveraging the structure of the trailer for support.

The Microhard radio coupled with the dish antenna is directly connected to the control consoles located inside the trailer. On top of the trailer a 7-element yagi antenna receives beacon telemetry from the satellites and is connected to a software defined radio located inside the trailer. A sensor package located on the dish mast is used for sensing trailer heading, pitch and roll and this data is fed to workstations located inside the trailer.

The complete system block diagram is located in Appendix B but a more readable version with the subsystem interconnections are shown in Figure 2.4.

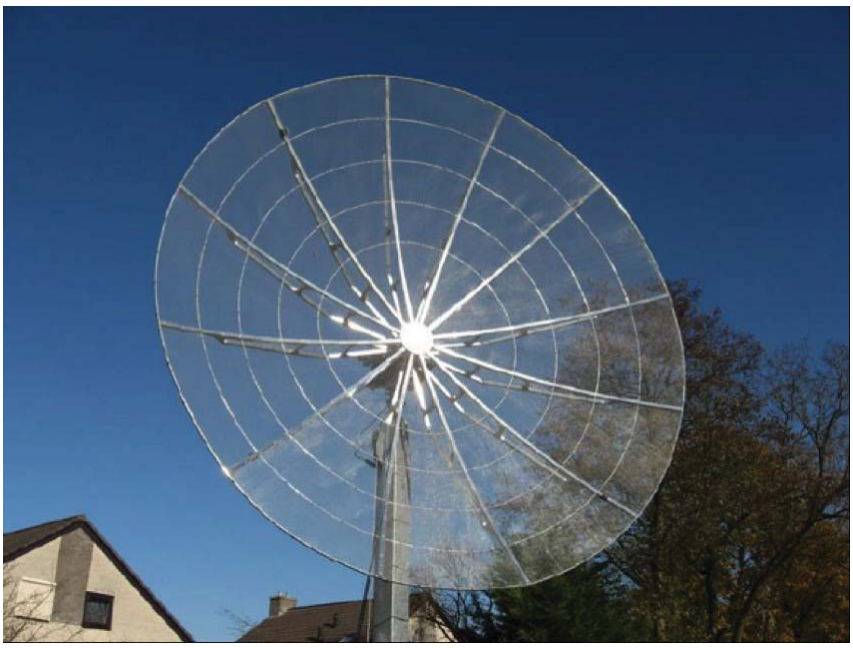


Figure 2.2: Three meter S band dish antenna from RF Ham Design

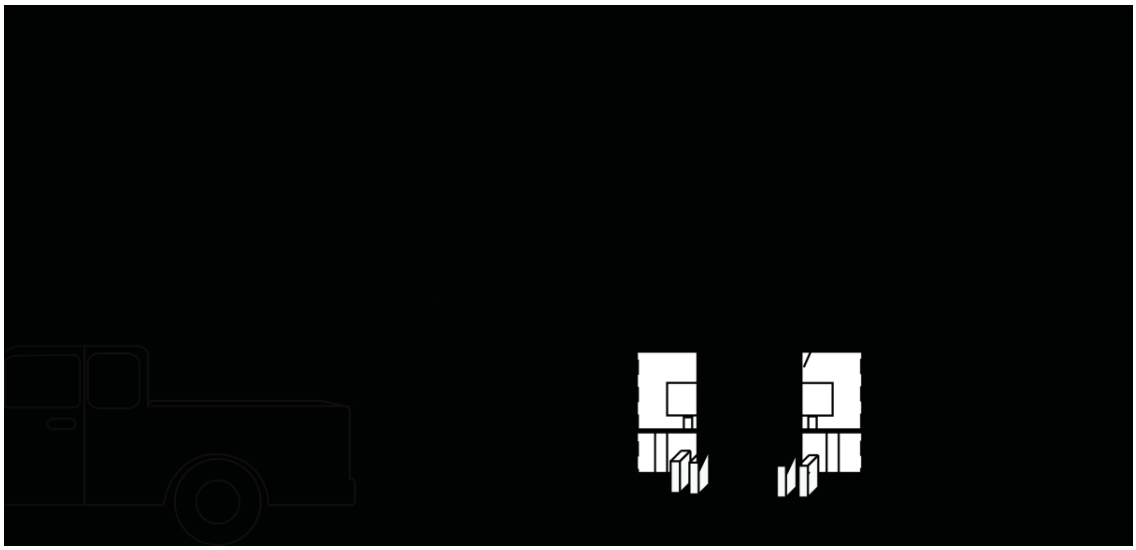


Figure 2.3: Cutaway view of the mobile satellite communication station

As seen in Figure 2.4, the mobile ground station makes use of six computer work-stations, two rotors each with a separate controller, two radios, two antennas, a sensor

package called the ArduPilot Module 2.6 or APM 2.6, and a network link. The S-band subsystem uses four of the six workstations for operation - analysis, S-band antenna tracking, S-band antenna data, and command workstation. Command and telemetry are handled through the command workstation and the S-band data workstation. The data is sent to a MicroHard radio attached to the dish antenna which transmits commands and receives satellite telemetry. The received telemetry is then stored on the command workstation database. The dish tracking is handled by the analysis workstation and S-band tracking workstation that connects to a rotor controller located inside the trailer. The rotor controller sends commands to the SPID RAS rotor attached to the dish to point toward the satellite.

The software defined radio subsystem uses three of these workstations for its operation - analysis, beacon antenna tracking, and beacon antenna data. The received data is handled by a piece of hardware called a Universal Software Radio Peripheral (USRP) and sent to the data workstation. The data is then processed on the data workstation and the packets are sent to the analysis workstation database. The analysis computer and tracking workstation cooperate to point the 7-element antenna by sending commands through an Arduino Uno micro controller connected to a rotor controller. The rotor controller sends commands to the GS-5500 rotor to point the antenna at the satellite as it passes overhead.

The auto-calibration subsystem consists of the APM 2.6 and the analysis workstation. The software located on the analysis workstation manipulates the tracking information stored on the local database to compensate for vehicle orientation before sending it to the tracking workstations.

The proposed mobile ground station was required to have performance similar to the RSL ground station according to the customer's requirements. To satisfy this, our station needed to include an S-band dish equipped with a Microhard radio, a 7-element antenna and amateur band reception hardware to replace the UHF/VHF antenna system, identical command consoles, and identical rotors and rotor controllers.

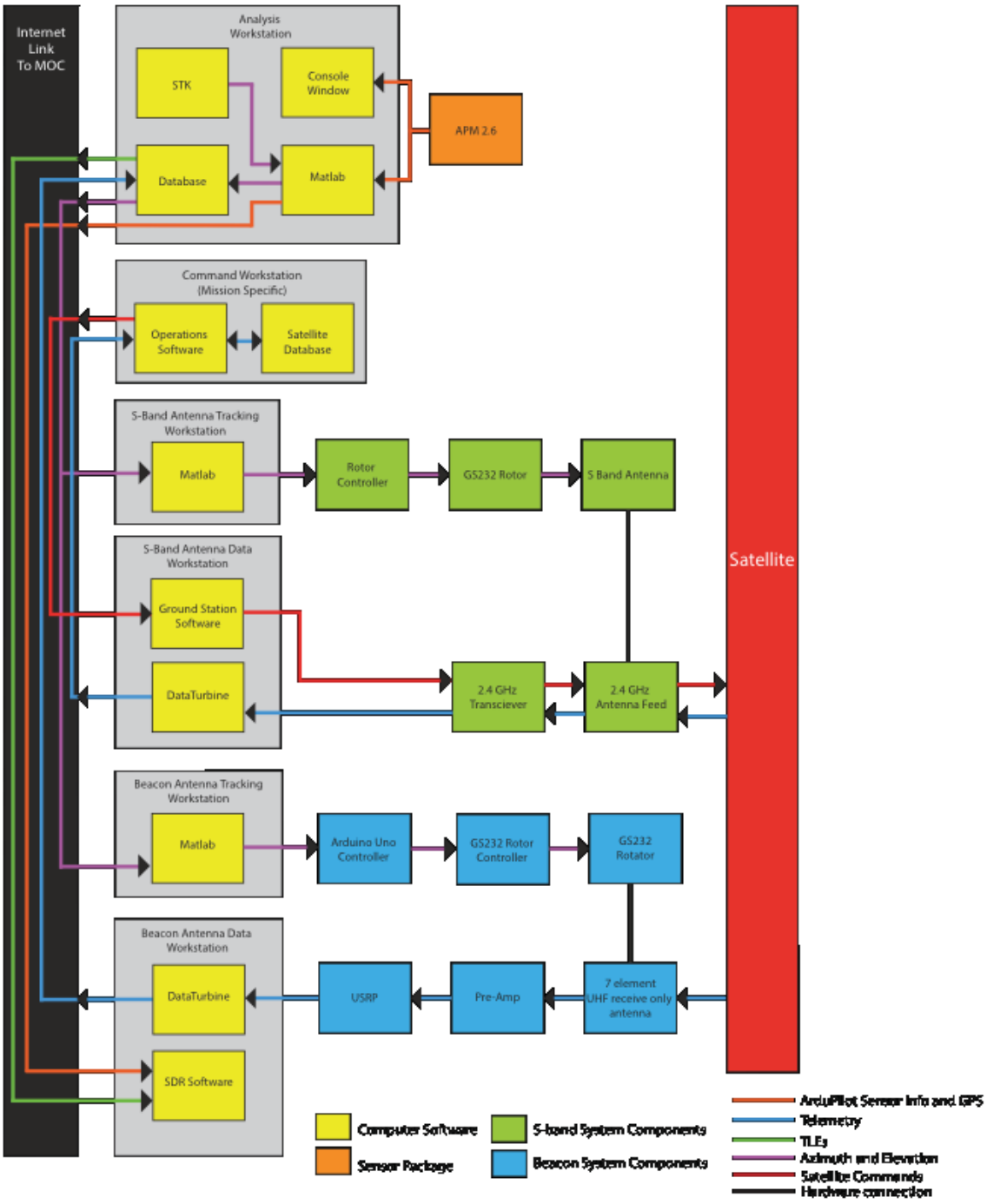


Figure 2.4: Mobile satellite communication station block diagram showing all major hardware components

4.2 Customer Needs and System Requirements

For our customer requirements we interviewed seven different customers, all having different project involvement. All interviews were done in person and asked the same list of questions except for one customer - Bruce Yost. First we interviewed Dr. Kitts, our primary customer contact, who was able to give us the bulk of our project requirements and several guidelines concerning the overall design. Mike Rasay was our second interviewee and gave us the most perspective on how to design the system for everyday use for operating the satellite. For our third interview we spoke with Thomas Adamek, who has had a lot of experience with setting up and modifying other Robotic Systems Laboratory systems. Our fourth interview was with a NASA employee named Bruce Yost who gave us insight to what influence a mobile station would have when contracting operators for nanosatellite missions. He was contacted by email and was given a separate list of questions that discussed NASA as a customer to the Robotic Systems Laboratory, rather than NASA as a customer to our design team. Our last three interviews were with Michael Neumann, Jake Hedlund and Steven Hiramoto who are all regular operators of the current nanosatellite control station. They provided advice on what the capabilities of the system should be to operate properly and conveniently while still allowing full mobility.

The primary goal of the customer review was to narrow the goals of the design as well as create more solid goal values for segments of the design. The interviews gave us a long list of possible features we could include in the system; however, not all of them were realized due to the short time frame we were given to complete the project. Accomplishing the requirements specified by Dr. Kitts, Mike Rasay, and Thomas Adamek were critical. Other features mentioned by other interviewees that do not overlap with these requirements were considered only after the critical requirements were met.

A few of the system requirements that were suggested in nearly all of the interviews were already specified in the preliminary design but will now be stated here for completeness. The system needed to be able to relocate to a desired location and perform

the same S-band command operations of the current Robotic Systems Laboratory ground communication station. It needed to also have similar performance of the current ground communication station where some performance was expected to be sacrificed for mobility. The largest driver of our project was to increase contact time with the satellite. A requirement of 60% increase in contact availability (for a given latitude) was set to ensure the mobile station would be worthwhile. Also there needed to be some mechanism to store the larger parts of the system, dish and antenna, during travel. The components of the system also needed to withstand any environmental elements they encounter.

One of the driving factors of our design was the setup time which includes dish antenna, calibration of the antenna, and control center setup. The setup times ranged from a maximum of three hours to a maximum of two days. Dr. Kitts, our primary customer, suggested that the system setup and breakdown time should be on the order of a day per process.

The minimum amount of people required to set up the system was also factored into the design due to the heavy pieces of hardware that needed to be assembled. From the interviews, it seemed that it was likely that two people would be setting up and operating the system. Consequently, the setup needed be safe enough for two people to assemble in under a day.

Another attractive feature mentioned in the customer reviews was calibration control for the dish antenna. The dish must be calibrated before contacting the satellite. This required some knowledge of the location and orientation of the vehicle. These attributes also needed to be accessed at a centralized location or possibly manually entered into software during a dish calibration procedure. This process needed to be done quickly and also efficiently to help decrease the total setup time of the system.

There are several other requirements that do not pertain to the mechanical design of the project. There should be some way to get Internet access, even if it is only temporarily and requires the operators to relocate. This way the time can be synced as well as updating the central data source. Having some form of living quarters, including a bathroom and

areas for sleeping is also a necessary for mobile ops. Due to the nature of the system it is likely that the station will be in the field for several days making it necessary for some way to house people when they are not operating the station. This requirement mostly puts a limit on the amount of space we are allowed to use in the acquired trailer since we will not be designing the living quarters for the mobile vehicle. A user manual for the system setup was also another requirement suggested for the system. This way new users can be trained properly and certified to set up the system. The system should also contain some form of power source that is off-grid. The power source should be able to power the entire system for approximately 8 hours.

4.3 Refined Project Concept

The following are the revised system requirements. They have also been tabulated in Appendix A.

Functional Requirements

- Increase contact availability by 60%
- Achieve about 5.1 dB link margin
- Create a method of collapsing antenna to 1.5m x 1.5m x .7m volume
- Decode 80% of beacon packets from O/OREOS

Nonfunctional Requirements

- Create a system that has the ability to relocate
- Design system to set up in under a day with minimum of 2 people
- Design system to break down in under a day with minimum of 2 people
- Use Robotic Systems Laboratory architecture
- Leverage existing RSL data paths
- Create a system rugged enough to withstand elements

4.4 Benchmarking Results

There are three systems that we explored that have similar functionality to the system we designed. The first similar system is the current SCU ground station that is used for nanosatellite operations. Since the station is at a static location, the dish antenna is not disassembled unless it needs maintenance. Our design has a dynamic location and the dish antenna will be broken down for travel and reassembled for satellite passes. One of the primary customer requirements was that the mobile communication station should have similar performance to the Santa Clara communication station and offer at least 60% of the pass availability for the same latitude. In order to operate similarly to the RSL, many of the same hardware components were used. The MicroHard radio used for S- band communication is identical to what the RSL uses for operations. Similarly the SPID RAS rotor and GS-5500 rotor are used in the mobile ground station and the RSL to ensure identical tracking capabilities.

In order to satisfy this requirement, a link budget analysis was done to ensure the link could be close with the satellite using a smaller 2.4 meter dish. Results showed that the station presented 71% of the contact availability assuming a connection would be established at the same link margin as the 3 meter dish the RSL uses. This 71% increase also assumed that our station would be static so the percent increase of availability could be larger given that the mobile station could be located at a more favorable latitude. For a nanosatellite mission with a polar orbit, this could yield a tenfold increase in contact availability. The dish antenna should also have a similar beam width to the Santa Clara antenna dish, approximately 3.2 degrees. The beam width for the 2.4 meter is approximately 4.0 degrees which allows for similar performance.

The second system that serves a similar function is GATR's 2.4 meter inflatable dish. This system uses a spherical ball to house their dish antenna, which is located at the center of the sphere. The concavity of the dish is established by creating a pressure difference between the opposite sides of the dish. The feed is located on the outside of

the sphere and allows for a consistent placement of the antenna feed in relation to the center of the dish. This system was particularly interesting because of its easy setup and small number of separate components. The setup time of this system is on the order of a few hours, which is significantly less than the 7 hour setup of the mobile ground station. Although the GATR system has a rapid setup time, it does not have the ability to track. The sphere is able to move approximately 10 degrees each direction in azimuth and is stationary in the elevation once it is set up. We lack the quick assembly time of the GATR dish, but compared to possible data rates of the GATR, the mobile ground station is superior due to its tracking capabilities.

The third system that we researched was Globecomm's Global Storm 2400 communication station. The antenna dish is housed on a separate trailer that is pulled by a vehicle. The dish can be set up in under an hour. This system has a greater range of motion than the GATR system with 30 to 90 degrees in elevation and 0 to 360 degrees in azimuth. Although the setup time is much faster than the GATR system, it is much more expensive (on the order of $100,000) and much heavier than the GATR antenna. Another downside to this system is that it requires a separate trailer to carry the antenna around. The cost the Global Storm is much greater than that of the mobile ground station making this design not optimal for nanosatellite operations.

4.5 Functional Analysis

The subsystems breakdown for our project is as follows:
Mobile Satellite Communication Station

1. S-band Parabolic Dish Antenna
2. Software Defined Radio
3. Auto-Calibration

4.5.1 S-Band Parabolic Dish Antenna

This system is the primary method of satellite communication. Commands to the satellite are sent through this system and satellite telemetry is also received through this system. The MicroHard radio paired with the 2.4 meter dish antenna and antenna feed are responsible for this communication channel, and a rotor is connected to the dish antenna to point at the satellite for optimal performance.

The inputs to this system are tracking commands and satellite commands. The outputs are satellite data and rotor movement. Satellite commands are sent from a computer to the MicroHard radio, and corresponding telemetry is sent from the satellite back to the Microhard which is transferred back to the computer. The rotor is also controlled by a separate computer which sends commands to the rotor controller that controls the rotor attached to the 2.4 meter dish antenna. The rotor then moves to the desired location.

There are several constraints on the S-band antenna in terms of the stowing and deployment mechanisms. The system needed to be safe to set up by a minimum of two people in under a day. The system needed to be able to fit on the tongue of the trailer and light enough for the trailer tongue to support the weight.

The performance requirement of this system is that it needs to increase contact availability by 60%. In order to do this the dish antenna must be able to obtain enough gain to close the link. The energy per bit to noise power spectral density ratio (eb/n0) helped us determine if our 2.4 meter dish would be able to accomplish the necessary gain. On average the link was established with O/OREOS using a 3 meter dish at an Eb/N0 of 20.6 dB. When using the equation 2.1 to determine the Eb/N0 for a 2.4 meter dish we found that 20.6 dB occurred at 24 elevation allowing for effective communication at a reasonably low elevation considering that this is a mobile application. More details of the link budget analysis can be found in the S-band subsection.

$$\frac{E_b}{N_o} = \frac{PL_lG_tL_sL_aG_r}{kT_sR} \tag{2.1}$$

4.5.2 Software Defined Radio

The software defined radio (SDR) is being developed to replace expensive pieces of hardware such as the ICOM IC-910 radio and to reduce cost for constructing a beacon station. The SDR also provides much more flexibility on which frequencies can be monitored simultaneously. The beacon antenna will be at the receiving end of the SDR and will be used mostly to monitor health and status data of the satellite. This is a supportive system for nanosatellite operations and is not vital for mission success.

4.5.3 Auto-Calibration

The auto-calibration subsystem ensures that the dish antenna and 7-element antenna are properly pointing toward the satellite. It uses a sensor package in the form of an APM 2.6 to determine location, orientation, and inclination of the mobile vehicle. This data is fed to the control consoles to offset the antenna rotators. Developing this system decreased the setup time of the system as well as automated a necessary step for performing operations.

The inputs to this system are heading, pitch and roll angles, GPS, second by second azimuth, elevation, and range information of the satellite's location in the sky from a frame assuming level ground and pointing north. This list of values comes from a software package called Satellite Tool Kit which has been used by the RSL for orbit propagation and satellite pass information. The output of this system is a corrected list of azimuth, elevation, and range information with respect to the frame of the trailer which takes into account the yaw, pitch, roll, and location of the vehicle.

The constraints of this system are that the information from the APM 2.6 must be accessible by all subsystems on the trailer, including those that are not a part of our project. A console window displaying sensor information on the computer attached to the APM 2.6 allows any system to use the data. The only performance requirement of this system is that it points toward the satellite with less than one degree of error.

4.6 Key System Level Issues and Tradeoffs

The key system level issues we've come to face are broken down to the satellite dish deployment and stowing mechanisms. Our driving requirements were the mobility and performance of the system. Finding a solution that can address quick assembly and disassembly while maintaining similar communication performance to that of the current RSL ground station. Design constraints, particularly with the RV, made it difficult to determine which design was best. Our design goals, determined by our Customer Needs Report, reduced the design options down to a handful of ideas: size reduction, dish case, panel assembly, flower collapsing dish, hinge collapsing, and telescoping-hinged arm. The goal is to develop stowing and deploying methods that can be carried out within a day (for both), can be performed safely by two people in the field, have a high performance similar to that of the current system, and have an intuitive setup process. By determining the pros and cons of each choice we've been able to perform a tradeoff analysis in order to select the most viable options for the design.

4.6.1 Size Reduction

Our first design option was to modify the size of our dish altogether. By reducing the size of the dish, additional modifications to the frame or mesh wouldn't be needed for storage, expanding options for deployment methods. Eliminating the size constraint of the trailer, we would be able to stow and deploy our satellite dish more easily. In addition, the reduced size could decrease the deployment speed . However, a smaller dish negatively affects the communication performance. Our final design did incorporate a size reduction moving from the standard 3 meter dish used by the RSL to the 2.4 meter dish. At the beginning of the project our customer the RSL expected a performance hit by attempting to implement a mobile solution.

4.6.2 Dish Case

This design option involved leaving the dish assembled and stowing it inside a case such that no dish modifications are needed and the dish can maintain its shape. A case would reduce the setup time greatly, make overall setup much safer for the operators, and is very cost effective. However, with the design constraints of the trailer we would have to make additional changes. Since the roof structure was not meant to carry heavy loads we would have had to reinforce the roof such that it can hold the weight of two people, the satellite, and the casing. With such design constraints, the stability of the vehicle during travel and operation came into question. Also, there are government issued vehicle clearance requirements that we must follow to have our vehicle cleared. Ultimately we did not decide to use this design due to safety reasons.

4.6.3 Panel Assembly

Panel assembly involves breaking the dish antenna into sections, each laying on top of one another for compact storage and light weight of individual components. The individual panel segments can maintain their shape reducing setup time of the dish itself substantially. The cons to this design are that it requires significant modification if the dish does not already come in separate panels. Our design does however incorporate the panel assembly. The dish antenna purchased came in four separate panels that are bolted together to create the 2.4 meter dish. The design is very compact fitting into a volume of 1.2m x 1.2m x .5 m which can be easily stowed inside the trailer.

4.6.4 Flower Collapsing Dish

This design involved a cranking method, similar to that found in a picnic umbrella, used to deploy the frame of the dish. This design requires very little assembling of the frame, is safe, has an intuitive setup, and requires less manpower than many of the other designs. However, extensive modifications are needed to be made on the mast and dish, there are issues with stability, and it requires a new dish surface solution. We did not end

up using this design due to the difficulty of the mechanism that would be required.

4.6.5 Hinge Collapsing

The Hinge Collapsing design is similar to the Flower Collapsing dish and caters to the height requirement issue. In this design the dish panels would fold in on themselves allowing for a more compact stowed volume. This design requires very little assembling, it's safe and intuitive, and requires less manpower altogether. However, like many of our designs, extensive modifications to the dish were needed. Also, we would require finding an alternative dish surface solution (that could potentially affect our performance), there are stability issues, durability issues, and possible maintenance needed on the hinges to extend life. We did not use this design due to the modifications needed.

2.6.6 Telescoping-Hinged Arm

This particular design seemed to be the most favorable of all our ideas. Using the base of the vehicle as a crutch the dish would stay assembled and a telescoping arm would move the dish upward. The issue of this design is that the arm needs to be in a location where the dish can stay attached. Inside the trailer was the most logical option but would require a hole cut in the roof of the vehicle. We also did not use this design due to the permanent modifications that would need to be done to the trailer.

4.7 Team and Project Management

4.7.1 Project Challenges

Difficult challenges we face throughout our project are a fair and flourishing team dynamic between each other and meeting deadlines. Communication is key to our project development both to meet deadlines and to remain on the same page. With different levels of involvement in the project it's important for us to communicate with each other to avoid the possibility of confusion and error in our design. Power distribution is crucial throughout our team. Equality among each other is essential to understanding the team goal before we begin working towards our success. To ensure the communication between our team

members, we continually agree to have open conversations on each decision that is necessary. Doing so creates an open, comfortable environment so that we can have our voices heard. Once a decision is reached on certain topics, we consult our advisor and clients. We are attempting to work as a unit to make each decision, not as individuals.

4.7.2 Budget

The budget for the project was $21,500. The entire cost of the system was $20,375. Details and breakdown of the budget can be seen in Appendix A.

4.7.3 Design Process

For the design process in our project, we began by individually assessing and brainstorming ideas and possible solutions to the various design challenges that our project had presented us. Once that step was completed as a team we came together and took turns presenting our ideas to each other. When all of the ideas were on the drawing board we began to list the pros and cons of each idea along with creating a tradeoff analysis. After this brainstorming session was completed we would take the results to our weekly meeting with our advisor where we conveyed our top ideas based off of our tradeoff analysis. Usually this meeting would unveil a couple ideas that we individually did not think of or a new problem that had not arisen until that point. We then would repeat this brainstorming process based on the new ideas from this weekly meeting.

4.7.4 Risks and Mitigations

With any project comes ethical ramifications. Particularly for our Mobile Ground Station, social and product ethics become risk factors to pay attention to. Careful consideration must be payed attention to the accuracy of our work. If any part of our Mobile Station isn't worked on extensively to the best of our ability we aren't ethically doing our job as engineers. It's our responsibility to design our station as best we can; any slack will create error and produce unethical work. However, along with such risks we must mitigate our options and design goals. There are several ideas we have produced to improve the current

station for SCU but if we reach too far we risk falling into unethical territory by speeding to get everything done on time. If we can manage to reduce our options of improvement to a reasonable amount and create a benchmark timeline with deadlines, we can ensure these risks could be avoided altogether.

4.7.5 Team Management

The ethical ramifications pertaining to our team management is very important primarily because of project is the beginning of a legacy project where future students will work on and improve the system we created. Therefore, the documentation, design work, and implementation of our system will need to be in the most refined version possible to allow the future success of the students that will try to improve our systems. In addition, we must uphold the highest ethical standards when dealing and working with each other as well as our advisor, sponsors, and our customers. This is very important to maintain because without any one of these groups our project would not be possible to complete or even attempt.

Chapter 5

S-Band Antenna

5.1 Overview

The S-band system is the most important system of the mobile ground station. It is responsible for downloading mission critical data and sending commands to the satellite via radio frequencies. The efficiency of the S-band system determines a large portion of the overall design success. One of the most important requirements from the customer was the 60% increase in contact availability which refers to the S-band radio link since the amateur band cannot download any mission data. Although a lot of the S-band system components on the mobile ground communication station are identical to the components used by the Robotics Systems Laboratory's static facility, some elements needed to be replaced or modified in order to sustain a mobile application. One of the major design choices made was to size down the dish from a 3 meter parabolic mesh antenna to a 2.4 meter solid surface aluminum antenna. Sizing down meant taking a significant loss in RF performance which will be discussed in later sections. Additional hardware parts were designed to increase mobility and support the dish antenna and mast.

5.2 Requirements

Since the S-band system is the most important system on the vehicle, several requirements were generated by the RSL to ensure project viability. A complete breakdown of the S-band communication system needs to be completed in under a 24-hour period. Likewise, the setup of this system has to be able to be completed in under a 24-hour period. Although

the RSL was willing to sacrifice performance for mobility, the station needs to offer 60% increase in contact availability. In order to achieve the contact availability requirement the system should have identical azimuth and elevation tracking rates to the RSL's static stations.

5.3 Options and Tradeoffs

In the early stages of development there were many approaches to the design of the S-band communication subsystem. Through our rigorous brainstorming sessions we determined many ways that the feed attachment design, mast structure design, and the support structure design could have been implemented. Each of these methods have their individual pros and cons as follows.

The design for the feed attachment has evolved from a static tripod that was fixed to the parabolic dish antenna. A negative aspect of this first design is that it does not allow the dish to be broken down into the volumetric restrictions of a 1.5x1.5x.5 meter volume. Furthermore, the rigidity of the static tripod prohibited ease of break down. Our next design option was a collapsible tripod using concentric tubing. Due to difficulties in manufacturability and fabrication this was abandoned when we could not find concentric pipes with minimal clearances to provide a snug fit. To get around this issue we altered our design to incorporate the use of pipes with the same diameter in two segments to complete each of the legs of the tripod. The two segments are connected with an adapter that was turned down the inner diameters of the pipes. This new design saved time in the manufacturing and fabrication processes, as well as savings in material cost.

The mast structure design mimics the design used on the static station at Santa Clara University. One of the design decisions adopted from the SCU mast structure is the hinge mechanism of the base plate connected to the mast structure. The hinge mechanism allows the mast to fold down completely horizontal allowing easy access to the rotor connection point. This hinging will also help decrease time during setup and breakdown operations. This was a key point in our design because this eliminated the use of extra manpower be-

yond the required two man maximum. In addition, this allows the operations teams to eliminate the use of ladders, which can be a big safety risk. The original design needed a ladder to top a 4 meter tall mast and required more that two operators to complete the setup and breakdown tasks.

Support structures in any design are crucial. The support structure involved with our project acts as an interface between the trailer tongue and the base plate for the mast structure previously discussed. This support structure design began with the idea of using two metal plates on the top and bottom of the trailer tongue and bolting them together creating a clamping mechanism. We went this route because it was required of us to not alter the trailer tongue by any means, which entails welding or drilling any hole in the tongue. When trying to implement this design we discovered that the clamped plate design was going to be too costly because the metal plate alone would cost a thousand dollars. We then approached this problem by making a frame out of 2" by 2" square tubing made of hot-rolled steel. This frame was then clamped using the original idea of creating a clamp via pressure with two bolts per point of clamping. This final design provided the rigidity, stability, and mounting point necessary for success during operations.

5.4 Detailed Design Description

5.4.1 Component Breakdown

The S-band system uses a lot of components used by the RSL in order to ensure tracking and data performance are as close to the static operation as possible. Figure 3.1 shows the hardware and software used to operate the S-band subsystem. The specification sheets for all these components can be found in Appendix D.

A picture of the RSL's dish antenna and components is shown in Figure 3.2. In the next few sections we will describe all the components used for the S-band system starting from the base support structure and ending with the RF equipment.

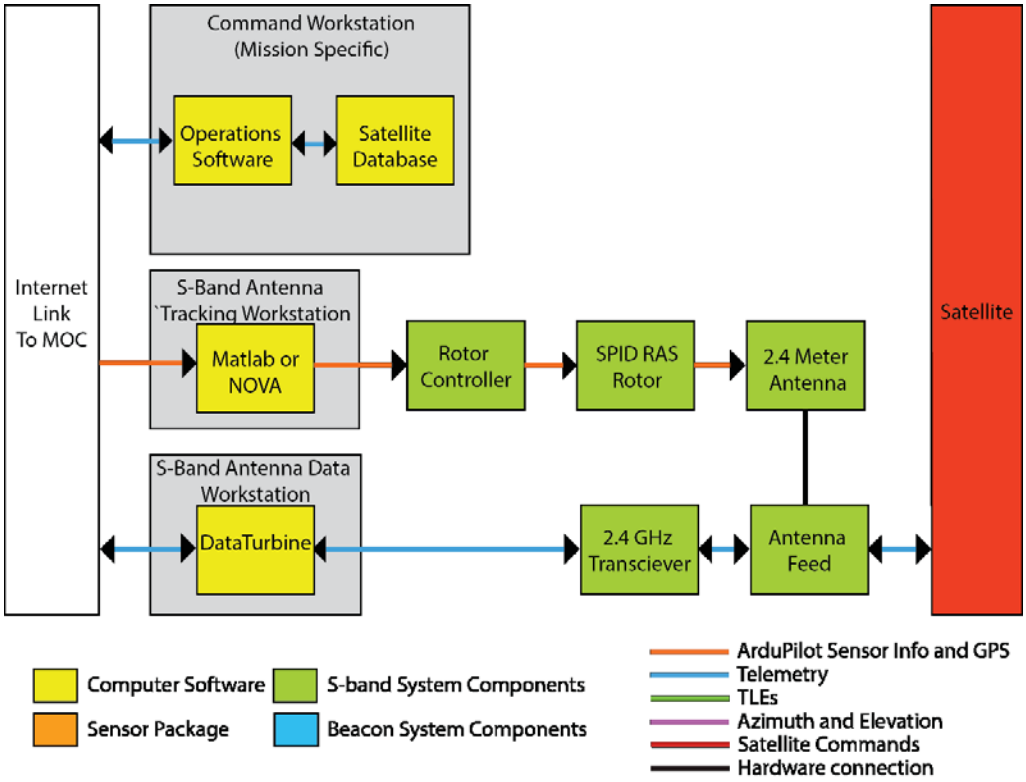


Figure 3.1: S-band System Block Diagram

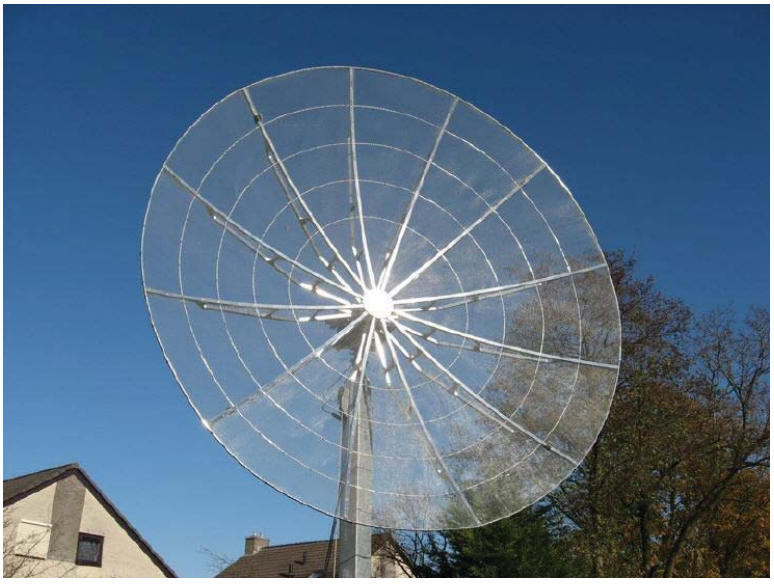


Figure 3.2: S-band Antenna Hardware Assembly

Support Structure

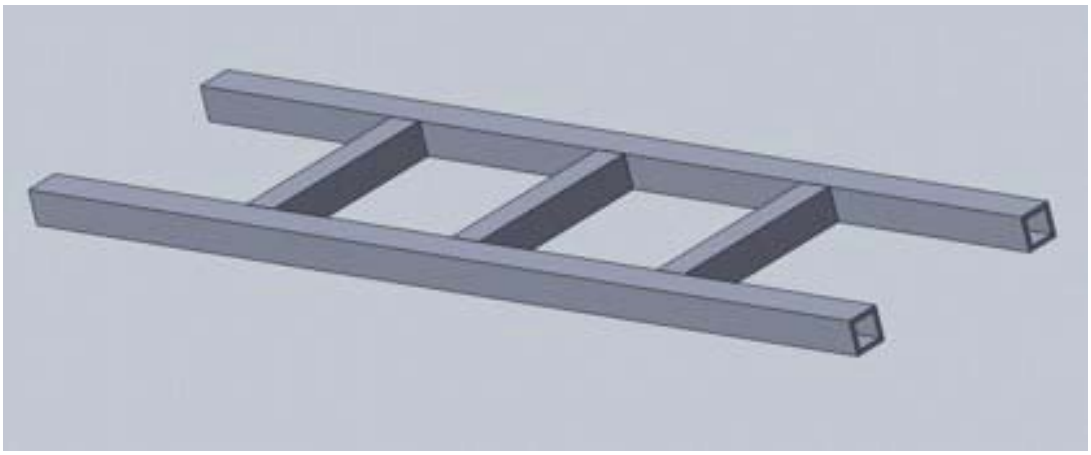


Figure 3.3: Assembled mast base support structure modeled in SolidWorks

The image above shows the assembled support structure of our design. The components are made up of hot-rolled steel square tubing and clamp onto the existing tongue of the trailer using another similar piece of hot-rolled steel. Figure 3.4 shows an exploded view of the mast base support structure. The structure is fastened using 1/2" bolts.

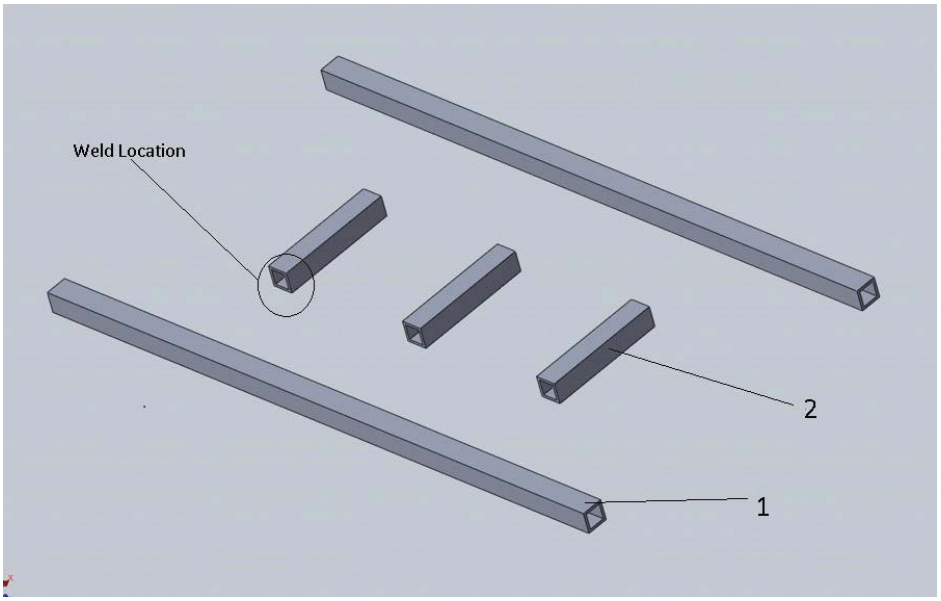


Figure 3.4: Exploded view of mast base support structure modeled in SolidWorks

The individual components and location of welds are identified in Figure 3.4 . The support structure's Component 1 creates the frame of the design to clamp onto the tongue. While the three smaller Component 2's support the frame in torsion and maintain a consistent shape of the system. 12-2" welds are applied to the top and bottom identified in Figure 3.4. As for hardware, we used 5/8" threaded rods and locking nuts to clamp the system together. This assembly has made our support structure fairly rigid; however, to support the vibration due to small angle corrections during satellite tracking, we added additional supports to the mast and guy wires to fasten to Component 1. Figure 3.5 shows the support structure assembled and attached to the trailer tongue.

Figure 3.5: Assembled Support Structure on Trailer Tongue

The next component of the assembly to be attached is the base plate using 1/2" bolts to the center of the ladder support structure in Figure 3.4.

Base Plate

The base plate chosen for our mobile communication station is identical to the one used by the RSL. The reason for this choice was the hinging mechanism that it offers. As seen in Figure 3.6, the base plate allows for the mast to fold down horizontally and allows easy access to the top of the mast. This feature is desirable in the mobile application because it increases the safety of assembly. The rotor and dish antenna are the heaviest components of the system, and being able to attach them to the mast at a lower height makes the setup process easier.

Figure 3.6: Antenna Mast Base Plate

Antenna Mast

Similar to the base plate, the antenna mast is identical to the RSL's because of its pairing with the base plate. The mast has been machined to fit the base plate adapters and therefore required no machining on the part of the design team. Also the mast provided enough strength to support the dish during windy conditions which will be discussed in the supporting analysis chapter. Figure 3.7 shows the antenna mast attached to the base plate which is fastened to the trailer tongue.

Figure 3.7: Dish Antenna Mast

SPID RAS Rotor

The SPID RAS rotor from RF Ham Design was chosen due to its ability to move a heavy dish as well as the tracking rates and ranges it provides. The 360 degree azimuth and 90 degree elevation range allowed for easy tracking with identical capabilities to the RSL's

static station. Also the rotor torque has proved able to operate a three meter mesh dish for several years without replacement. The RAS rotor controller was also chosen so that a new controller would not have to be developed. Figure 3.8 shows the SPID RAS rotor and rotor controller used for the mobile communication station.

Figure 3.8: SPID RAS Rotor and Rotor Controller

2.4 Meter Dish

The 2.4 meter dish from Worldpromotion Inc. was chosen to increase the mobility of the system. The 3 meter dish used by the RSL is also more difficult to setup due to the late number of components. The 2.4 meter dish is a solid surface aluminum dish that is made up of four separate panels that are bolted together as seen in Figure 3.9.

This dish was chosen not only for its reduced size but also because of the panel assembly. It dissembles to a very small volume which helps us achieve the stowed volume requirement. Ultimately it allowed us to implement one of our stowing methods without doing extensive modification to the original dish. The setup is also very intuitive, requiring only that the user bolt the panels together allowing the dish surface to maintain its shape.

Figure 3.9: 2.4 Meter Solid Surface Aluminum Dish Antenna

Dish Adapter

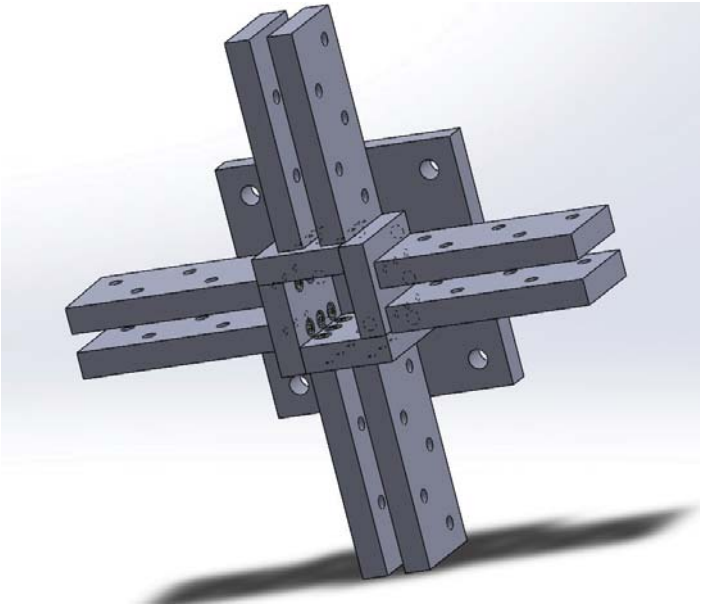


Figure 3.10: Rotor to dish attachment modeled in SolidWorks

Figure 3.10 shows an assembled image of the dish adapter used to fasten the dish onto the rotor plate that controls azimuth and elevation of satellite tracking. The design, similar to the Feed Design, is composed of Aluminum 6061 alloy. However, for the adapter design we bumped up the material thickness of the aluminum to half an inch to support the load

from the dish. Our design is composed of three components not including hardware to fasten the pieces together and was largely restricted by an available space of 2.25” x 2.25” to attach to the dish itself. The components are shown below:

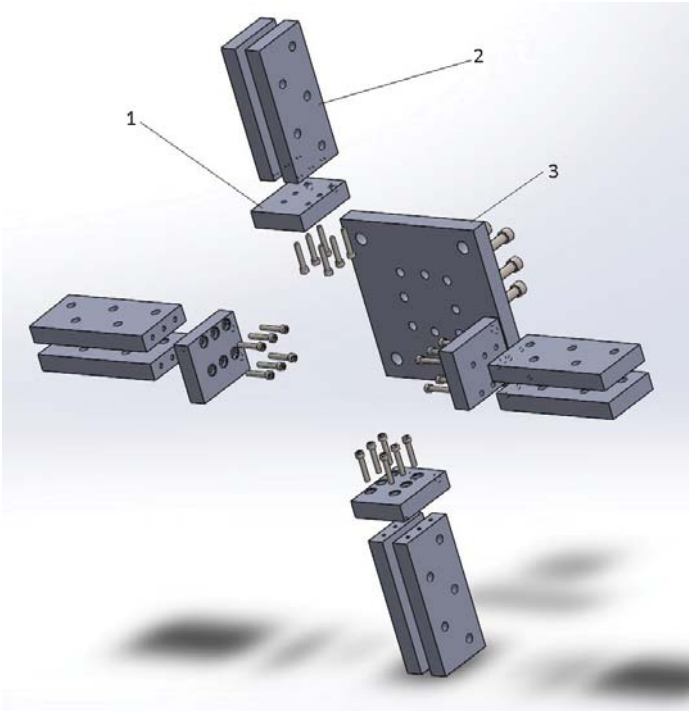


Figure 3.11: Exploded view of rotor to dish attachment modeled in SolidWorks

Here we see the labeled Components. Component 1 is our adapter wall attaching to the base (Component 3) and its corresponding fins (Component 2). Assembled together, the adapter walls create a 2.25” square in order to fit perfectly into the available space on the dish. Component 2, the adapter fins, attach the adapter assembly to the dish fins. To maximize the strength in our attachments we used 1/4” bolts to attach Component 2 to the dish and 6-32 screws, with counterbores, to attach Component 2 to Component 1. Finally Component 3 attaches the entire system to the Rotor plate. Using 1/4”-28 bolts, with counterbores, Component 3 attaches to Component 2. The final attached assembly is shown in Figure 3.12. Figure 3.13 shows the adapter that was machined according to the specifications. The adapter is attached to the rotor hub which attaches to the rotor during operations.

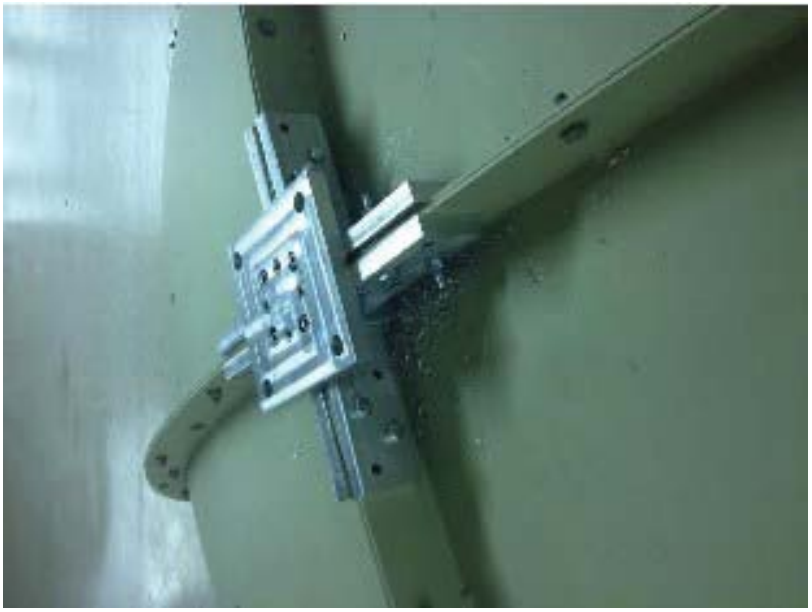


Figure 3.12: Photo of the rotor to dish adapter connected to the dish antenna

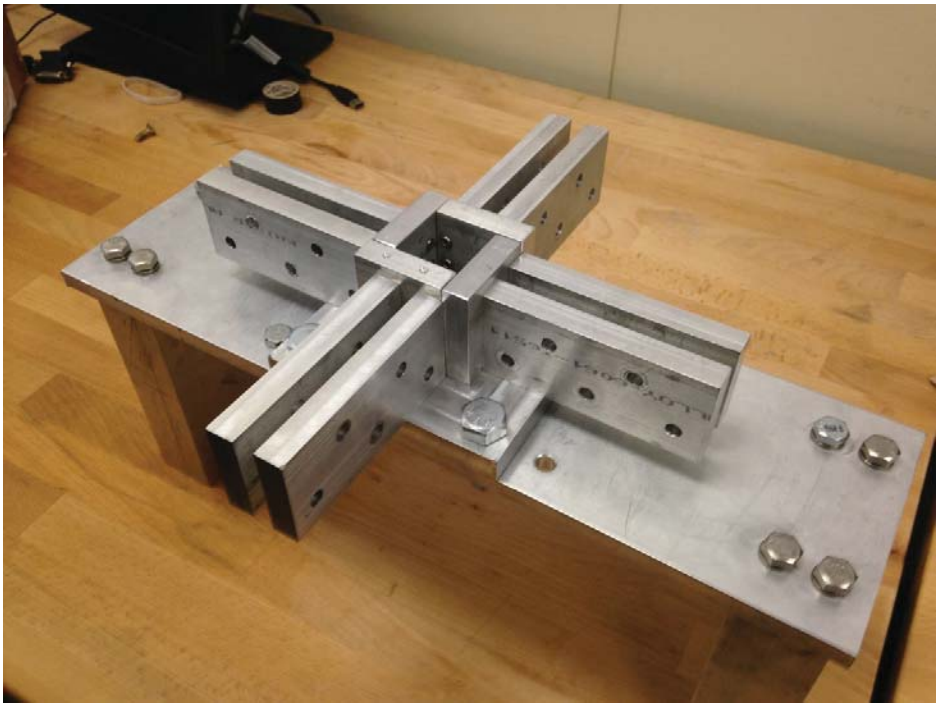


Figure 3.13: Dish Adapter Connected to Rotor Hub

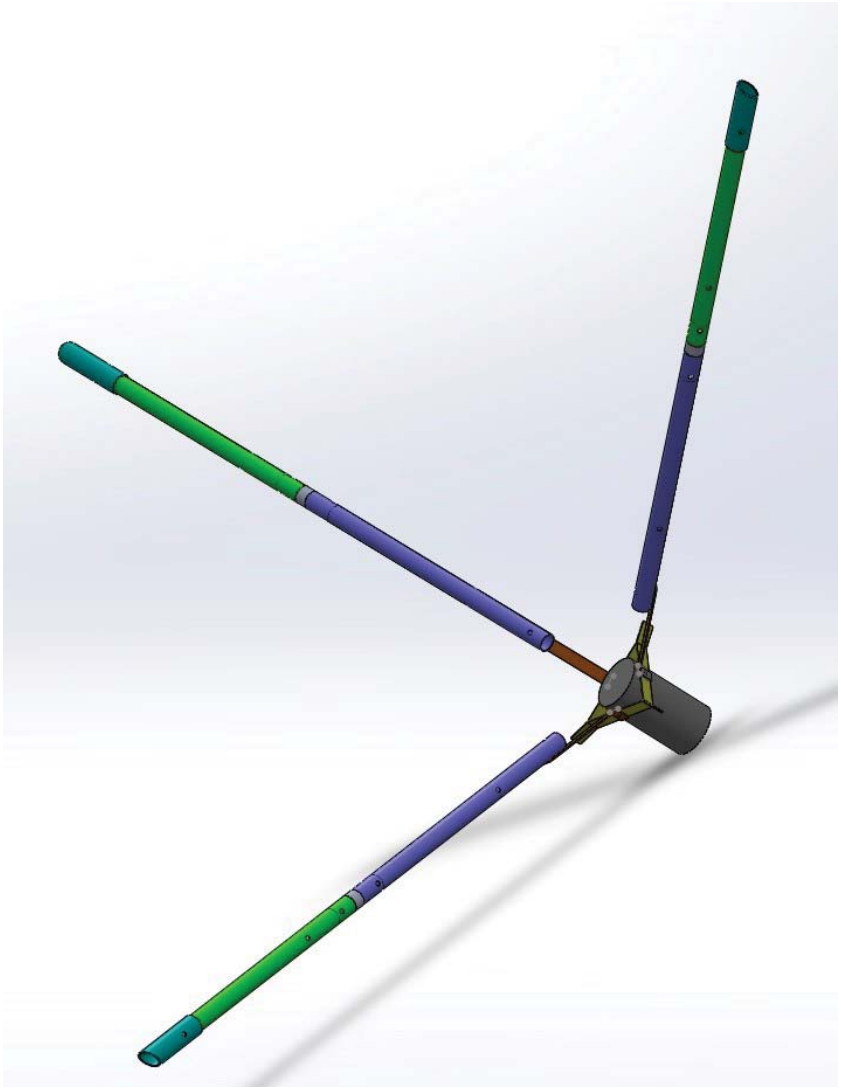


Figure 3.14: Feed structure assembly modeled in SolidWorks

Figure 3.14 shows the assembled Feed Design we've developed over the course of the year. The design uses Aluminum 6061 alloy due to its strength properties, light weight, and ease of manufacturability. The different components of the assembly are shown in unique colors in the image. Below is an exploded view of the design:

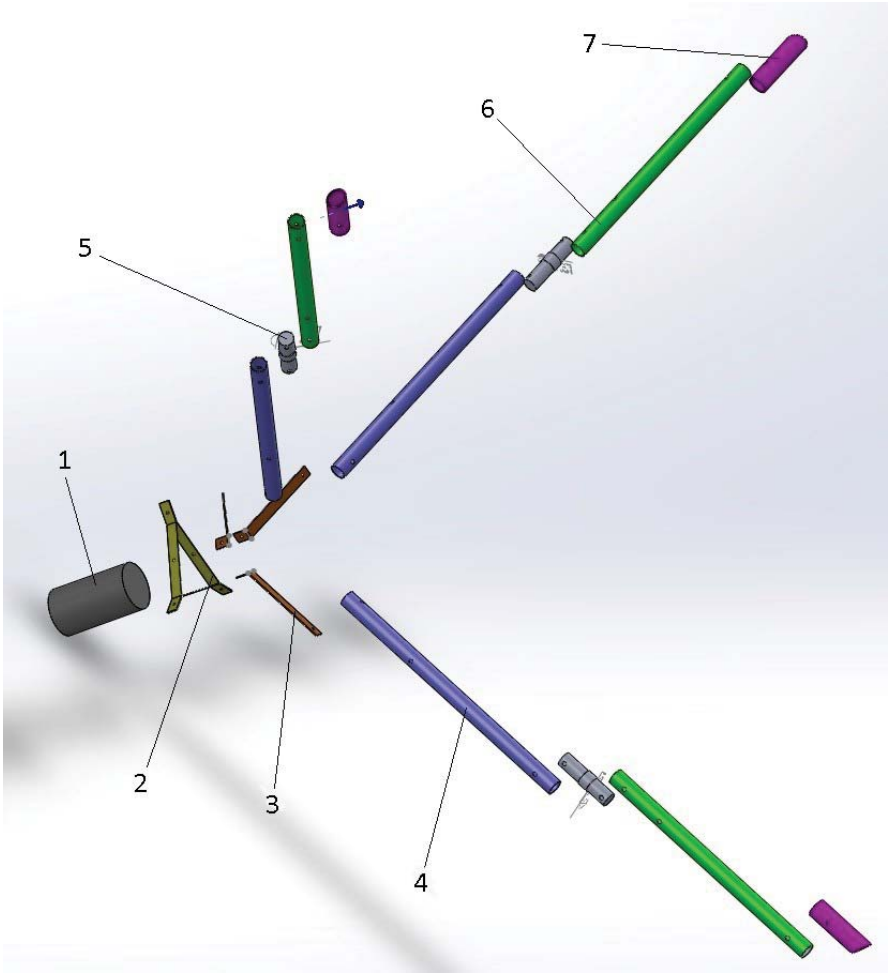


Figure 3.15: Feed structure exploded view modeled in SolidWorks

Beginning in gray, Component 1 represents the cylindrical feed used to receive data reflected from the parabolic dish attached to Component 7 (in pink). With fixed dimensions, Component 2 was designed to fasten the feed onto the assembly. We used low-carbon steel 14 gauge sheet metal due to its ease of manufacturability to hold the feed in place, creating a pressure-hold between its two other counterparts. Component 3 is composed of the same low-carbon steel sheet metal to attach the sub-assembly to the tripod-arm assembly beginning with Component 4 (in blue). Components 4, 5, 6, and 7 are made of Aluminum 6061 alloy as mentioned earlier. Components 4 and 6 have similar diameters of 1.5" and Component 5 is uniquely machined to fit to its corresponding Feed-arm, due to possible

warping of the material. Similarly, Component 7 is machined to fit Component 6 with a tolerance of within 50 thousandths. Figure 3.16 shows the arm that was manufactured.

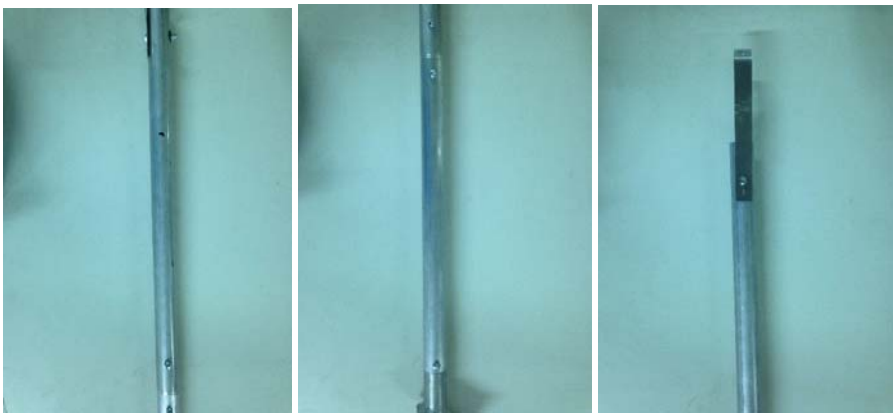


Figure 3.16: Antenna Feed Structure Arms

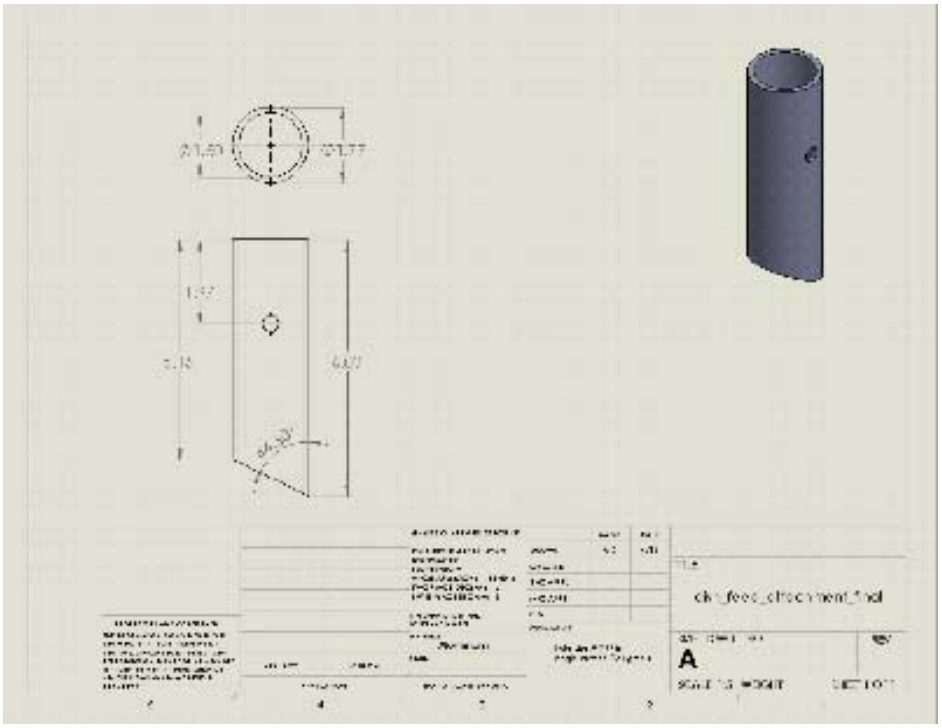


Figure 3.17: Feed structure attachment adapter to the dish rim modeled in SolidWorks

As for manufacturing most components were simply cut to size within tolerance and had holes drilled for the fasteners of the assembly. One component, however, required

more attention. To the critical distance between the feed (receiver) and the dish surface, we needed to calculate the exact distance and angle necessary for the feed assembly. Using basic trigonometry and measurements we determined the angle needed to be 64.3 degrees, shown in Figure 3.17. Figure 3.18 shows the manufactured dish attachment.

Figure 3.18: Antenna Feed Structure Dish Attachment

Antenna Feed Receiver

The antenna feed receiver from RF Hame Design is connected to the MicroHard radio and allows for it to transmit signals on the dish antenna face for satellite communication. The feed used for the mobile ground station is a left-handed circular polarization antenna feed that is identical to the one used by the RSL. The RF data path was not redesigned for the mobile communication station because it did not add to the mobility of the final product. This is why all of the RF components, excluding the dish, were the same model as the one used by the RSL. Figure 3.19 shows an image of the dish antenna feed.

Figure 3.19: LHCP HELIX Dish Antenna Feed

MicroHard Radio

The MicroHard Radio designed by Microhard Systems Inc. is responsible for all of the data download and transmission. It is the main method of communication for the mobile communication station and its performance is highly reliant on the equipment that it is attached to. The MicroHard radio is connected to an antenna feed which broadcasts the signal onto the face of a dish antenna. The radio waves bounce off of the dish antenna and move in the direction the dish is facing. The shape of the dish antenna will affect how these radio waves will be transmitted mostly determining power and beam width. Figure 3.20 shows an image of the MicroHard radio.

Figure 3.20: MicroHard 2420 Radio Used For Radio Frequency Communication

Software and Software Architecture

Although the software architecture of the RSL was maintained for this project it is important to note that the software used in the system plays a large role in successful com-

munication. Figure 3.1 shows the software packages used on the specific workstations. The ops software designed by Robotics System's Laboratory employees sends the MicroHard commands to be transmitted to the satellite and is also responsible for storing data that is received from the satellite into the database for future analysis and data product delivery. DataTurbine is the software that runs in-between the MicroHard radio and the ops software, creating a data path to maximize system modularity. Matlab and Nova are used for satellite tracking and are connected to the SPID RAS rotor controller to send second by second tracking commands to the rotor. Again, the modular design of the workstations and their software was adopted from the RSL's model since there was no obvious mobility benefit from a software overhaul.

5.5 Supporting Analysis

In order to verify that our design was stable enough to be used in the field, we performed a finite element stress analysis on the assembly using SolidWorks. We tested wind loads of 14 m/s, which is the highest wind speed that we would expect to operate at. We did two separate analyses to ensure the sturdiness of our system. The first analysis was testing the stresses on the dish mast with the wind load applied to the face of the dish. The second test was on the unattached dish antenna. The surface integrity of the dish is important to communication and if it yields under wind loads then it may affect the communication performance. Table 5.1 shows the material properties used for the dish mast during the finite element simulation.

Table 5.1: Mobile Communication Station S-band Mast Properties

Model Reference	Properties		Components
	Name:	Galvanized Steel	SolidBody 1(Boss-Extrude5)(Mast-1), SolidBody 1(Revolve2)(dishv2_90degree-2), SolidBody 2(Cut-Extrude2)(dishv2_90degree-2)
	Model type:	Linear Elastic Isotropic	
	Default failure criterion:	Max von Mises Stress	
	Yield strength:	2.03943e+008 N/m^2	
	Tensile strength:	3.56901e+008 N/m^2	
	Elastic modulus:	2e+011 N/m^2	
	Poisson's ratio:	0.29	
	Mass density:	7870 kg/m^3	
Curve Data:N/A			

Name	Type	Min	Max
Stress1	VON: von Mises Stress	5.41353e-007 N/m^2 Node: 1663	7.04822e+007 N/m^2 Node: 5016

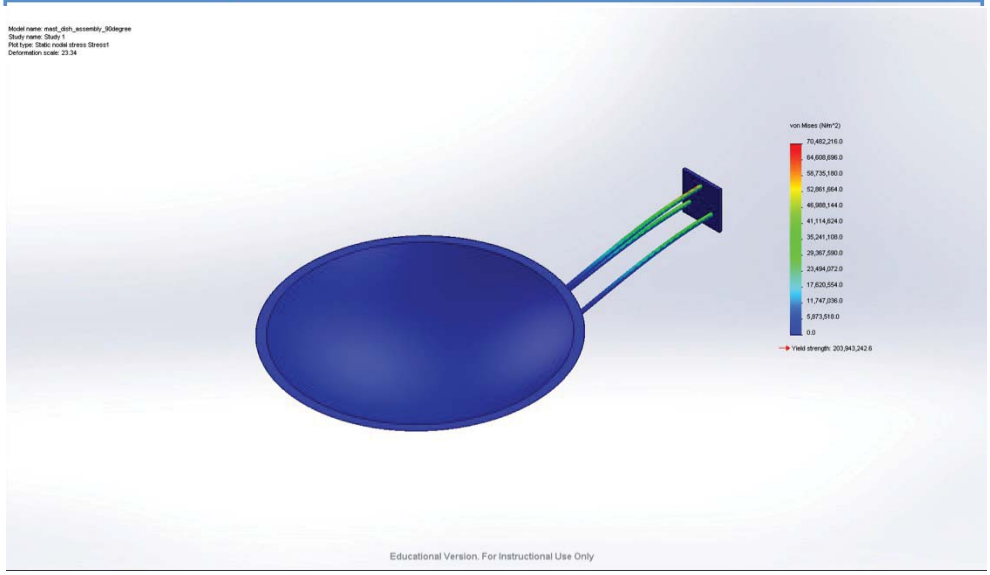
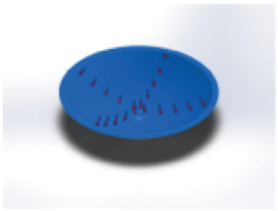


Figure 3.21: Finite Element Stress Analysis on the S-band Antenna Mast

Figure 3.21 shows the model result of the analysis. The maximum stress is seen occurring at the base plate with a maximum stress of 7.04822e+007 N/m^2 which is well below the maximum yield strength of galvanized steel.

Table 5.2 shows the material properties used for the dish during the finite element simulation.

Table 5.2: Mobile Communication Station S-band Dish Properties

Model Reference	Properties		Components
	Name: Model type: Default failure criterion: Yield strength: Tensile strength: Elastic modulus: Poisson's ratio: Mass density: Shear modulus: Thermal expansion coefficient:	1060 Alloy Linear Elastic Isotropic Max von Mises Stress 2.75742e+007 N/m^2 6.89356e+007 N/m^2 6.9e+010 N/m^2 0.33 2700 kg/m^3 2.7e+010 N/m^2 2.4e-005 /Kelvin	SolidBody 1(Revolve2)(dish)
Curve Data:N/A			

The stress analysis done for the dish antenna shown in Figure 3.22, shows a maxi-

mum stress of 229461 N/m^2 which is well below the yield strength of 1060 aluminum.

Name	Type	Min	Max
Stress1	VON: von Mises Stress	3817.59 N/m^2 Node: 7765	229461 N/m^2 Node: 8121

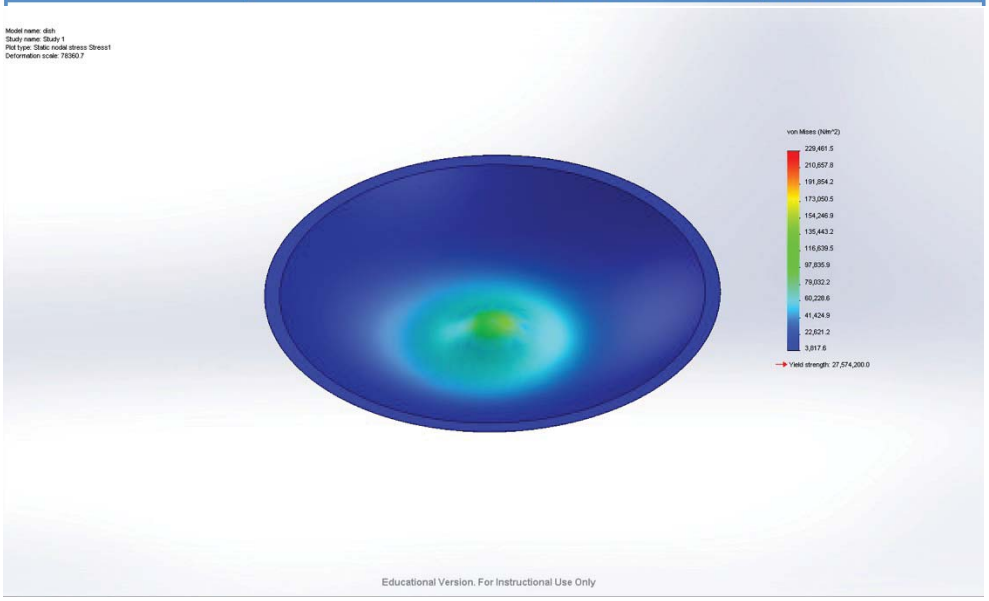


Figure 3.22: Finite Element Stress Analysis on the S-band Dish Antenna

5.6 System Progress and Future Testing

There was significant progress on the S-band subsystem. The team designed new methods for attaching a 2.4 meter dish to the standard rotors used by the RSL. The trailer tongue was modified in a way to support maximum mobility and safe setup for two people. A quick method for attaching the antenna feed to the dish antenna was also designed to reduce the amount of setup time required for the S-band system. Despite the numerous accomplishments on the subsystem, there were some actions that were not completed.

We were not able to fully assemble the S-band system and test against the O/OREOS satellite due to time constraints. The hardware in order to support the mobile application has all been designed and manufactured and the component connections have been tested and verified. The next step of the design process should be commanding a real satellite. This will test that actual increase in availability and the performance of the dish antenna compared to the three meter dish used by the RSL.

Chapter 6

Software-Defined Radio

6.1 Overview

The team sought the creation of a functional software defined radio capable of receiving and decoding beacon packets with an acceptable ratio of decoded packets. The Software Defined Radio subsystem aggregates several functions within the mobile ground station.

Software Defined Radio (SDR) is a radio communication system where components typically implemented in hardware (such as mixers, filters, amplifiers, modulators, demodulators, and detectors) are instead implemented via software on a computer. While the concept of SDR is not new, the rapidly evolving capabilities of digital electronics allow changes of functionalities and applications that theoretically would be very demanding if we were to change the hardware.

A basic SDR system may consist of a personal computer equipped with a sound card, an analog to digital (A/D) converter, preceded by a radio frequency front-end where down conversion and analog filtering occurs. Significant amounts of signal processing tasks are supplied to a general purpose processor (GPP), FPGA or a DSP module instead of being implemented in hardware. This design produces a radio that can receive and transmit throughout different radio protocols or waveforms based solely on the software used.

The SDR software performs the demodulation, filtering (both radio frequency and audio frequency), and signal enhancement such as equalization and binaural presentation.

Its applications include every common amateur modulation: morse code, single sideband modulation, frequency modulation, amplitude modulation, and a variety of digital modes such as radioteletype, slow-scan television and radio packages.

In order to make the current ground station mobile, the system had to be flexible, versatile, easy to physically move to the Recreational Vehicle and reliable. The architecture of the Software Defined Radio fits our purposes since it has a converter circuit frequency quadrature (hardware), which converts the frequency of the RF signal for a sufficiently low power intermediate frequency IF to be processed by a computer sound card or other A/D (analog to digital) suitable in two lines, called I and Q.

In the next step, an open source software (GNURadio) allows the signal to be processed and the I and Q signals coming from the hardware are digitized by the A/D converter of the sound card. This software performs the appropriate mathematical combination of the I and Q signals to reject the unwanted image frequency existing in frequency conversion, and then performs the demodulation of the signal (such as AM, SSB, FM, and DRM).

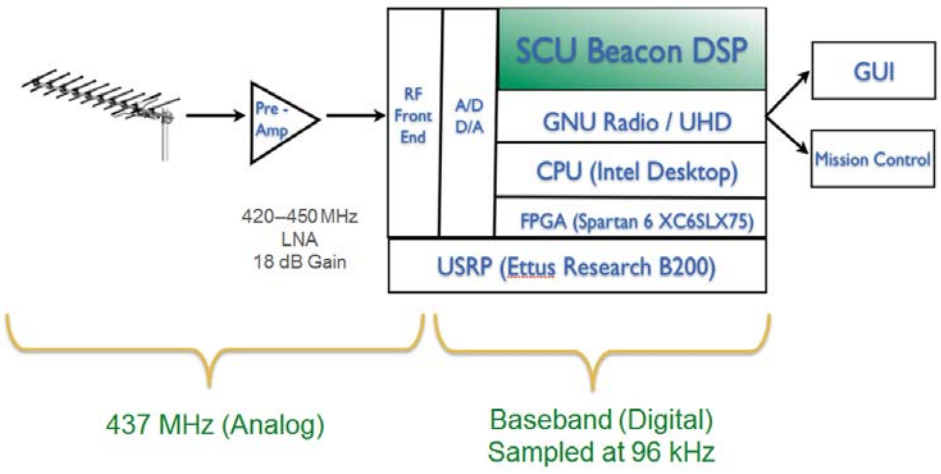


Figure 4.1: Software-defined radio architecture

The Figure 4.1 represents an overview of the Software Defined Architecture. Cost, reliability and computational cost were the most weighted factors upon the decision of such system layout.

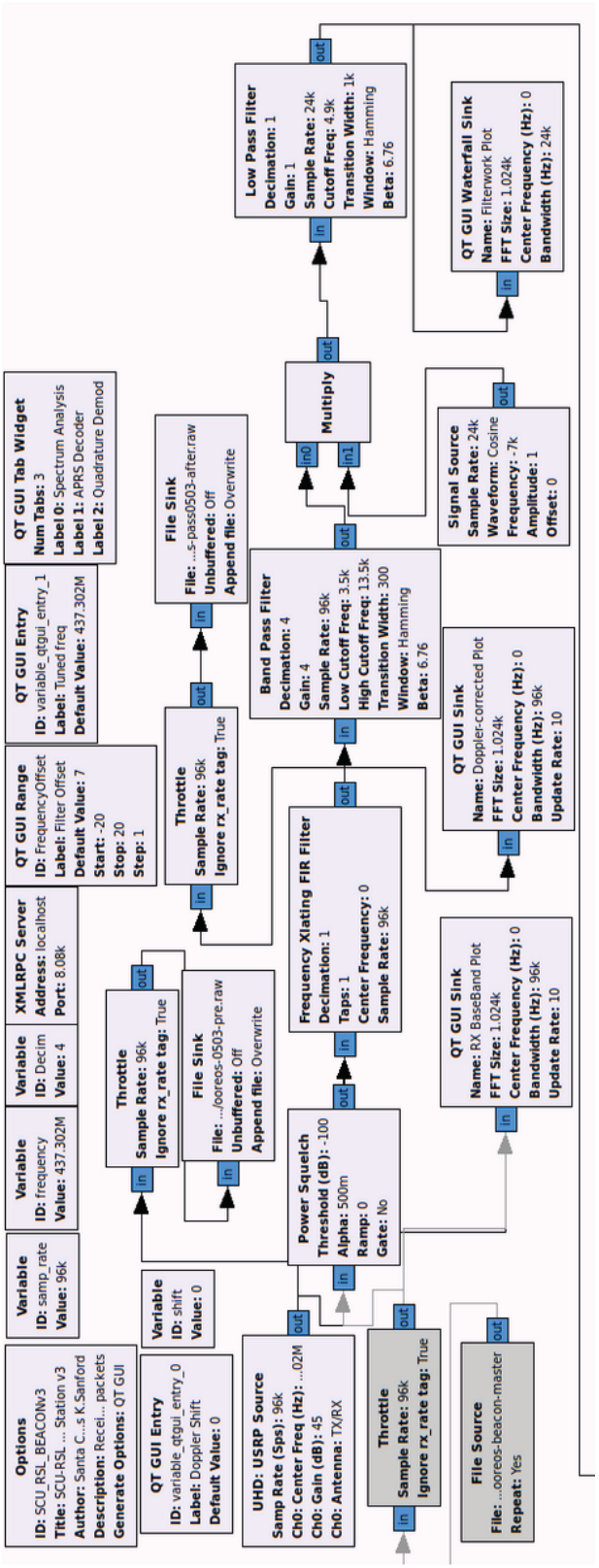


Figure 4.2: Software-defined radio filter diagram for the mobile communication station, output from low pass filter decimation block

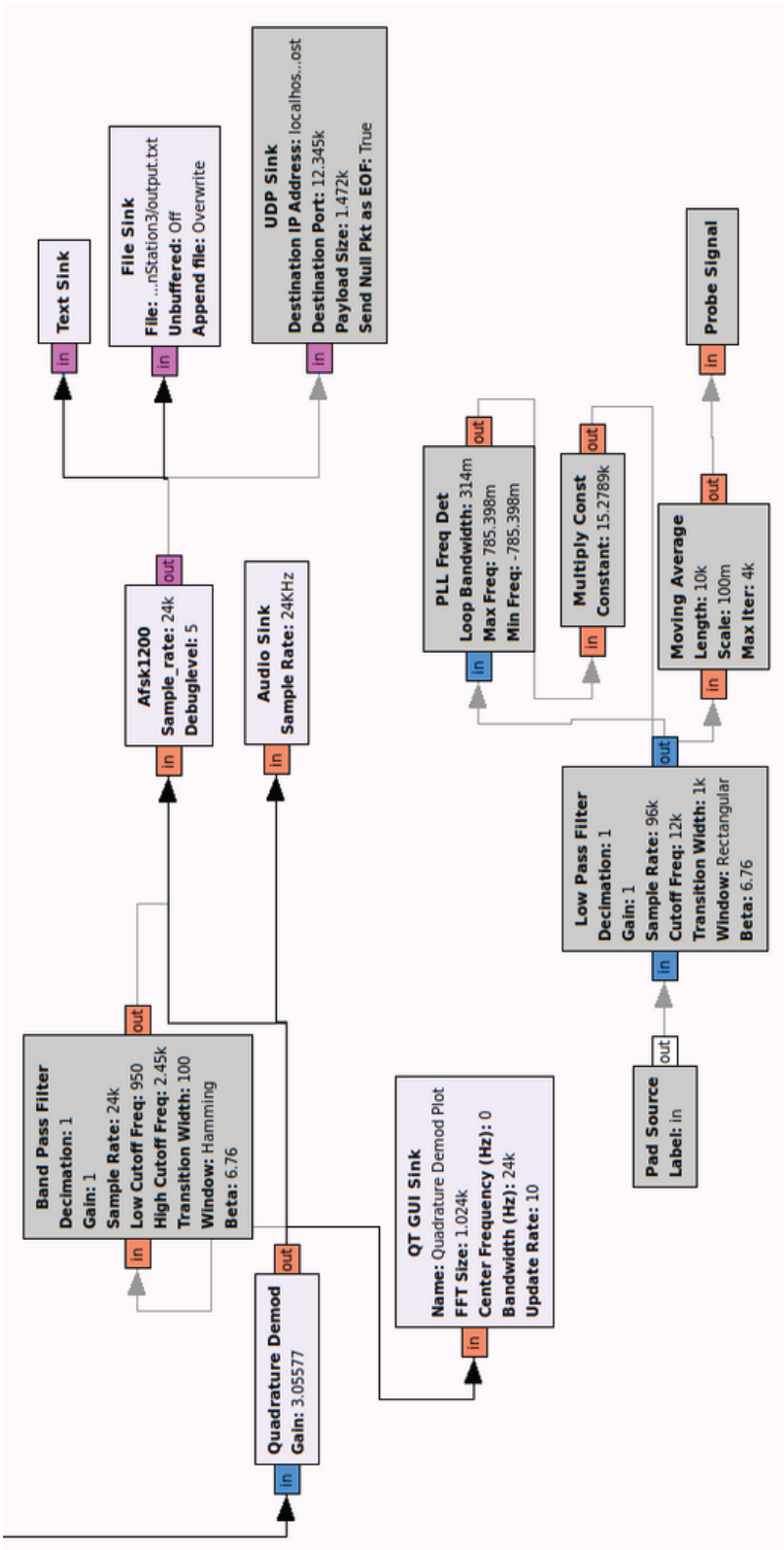


Figure 4.3: Software-defined radio filter diagram for the mobile communication station, input from low pass filter decimation block

The signal of 420 to 450 Mhz comes through a 7 element Yagi antenna with a gain of 13 dBi. Then it gets amplified by 18 dB in the Low Noise LNA amplifier. Finally the electrical impulse reaches the USRP 200 where the analog circuitry performs the down-conversion of the signal as well as the analog low pass filtering task. The board is designed on a stack. On the lower level of the stack, the FPGA (Spartan 6 XC6SLX75) performs the heavy duty digital signal processing tasks. In an external environment, a CPU houses the graphical user interface where the user can interact with the software. Higher on the stack, there is the GNU Radio platform, an open source framework similar to Simulink which allowed us to develop, test and deploy our radio raw design represented by the block shaded in green.

6.2 Requirements of the Software-Defined Radio

The project was funded by NASA and the Robotic Systems Laboratory at Santa Clara University. As final customers, they required the system to have a friendly user-interface. The operational cost of training a technician to maintain and manipulate the software is acceptable if we were to put some thought on how the design of the graphical user interface would look and appeal to the general undergraduate engineers. The HAM radio installed in the static station of Santa Clara University compensates for the Doppler shift effect with external software and hardware. This feature had to be implemented as a proof that the software radio technology surpasses the limits of conventional radios once the correction could be done inside the software developed in the USRP. This product had to be designed with the minimum physical space possible since the mechanical apparatus in the Recreational Vehicle takes 60% of the space.

On the operational practical perspective, our customers required a maximum set up time of 10 minutes at the moment the trailer becomes static. This is extremely important since satellites normally have a 10 minute window overhead in relation to the position of the parked RV. The idea is to arrive at the final destination at least 10 minutes early to make sure the system is running correctly.

In the technical side, our system had to cope with a minimum of 80 % success of decoded packets. This means that the digital processing tasks should be optimized to have as low a computational cost as possible so the application can run in real time with the minimum chances of failure. The ability to record passes was an additional requirement in our system. This allows the operator to simulate the pass of a satellite and perform fine tuning such as some sort of shift in the center frequency of the beacon or temperature change causing the crystal of the system to oscillate more than desired.

Today, NASA deals with an average of 500 satellites in different orbit trajectories. A system which could accommodate multiple satellites is crucial to run a smooth operation. Our system had to accommodate at least 20 satellites simultaneously being able to compensate for the Doppler shift which also has to be tracked live. The operator as well as the RSL mission control center ideally wants to be notified of new packets and the faster the information is distributed the more competitive advantage our product brings in relation to alternative solutions. Our product had to come up with new and innovative ideas to share the decoded packets preferably through a network connection or through a TCP/IP port.

The cost diluted in purchasing new hardware and cables had to be accounted as an important trade-off. A cost analysis was done to assure that new functionalities and capabilities would increase the performance and productivity of the laboratory overall.

6.3 Options and Tradeoffs Summary

In our design process there were many different solutions we considered in order to solve our problem. All had certain positive aspects and drawbacks associated with them. Through research and as well as testing, we tried to find the optimal system that performed all necessary tasks within a reasonable price range. Here are some of the comparisons we made throughout our design process.

Front-end tools for a software defined radio system consists of the hardware that receives the signal and converts it into a bit stream that will then be transmitted to a pc or micro-controller via a bus such as USB or LAN. In our design process we considered a few

different front-end design configurations and compared the advantages and Disadvantages of those systems.

6.3.1 FunCube Dongle

When our team joined the satellite project the Robotic Systems Lab had been experimenting with a radio receiver for software implementation called the FunCube Dongle. To this point the FunCube had typically been used by hobbyists in amateur radio applications.

Some of the most attractive features of the device is how simple the hardware setup is; it has two connections, an in using the standard SMA female antenna port and an out using a USB male connection. It was also extremely cheap costing approximately $100 dollars. It does not require the installation of any driver software and is fully supported by GNU Radio. All these features and more is the reason this is the most popular device used in SDR design at the current moment. Because of its popularity it routinely receives additional support with GNU and will never become obsolete. It was a great tool for troubleshooting our design blocks since it was a very simple and never failed.

Figure 4.4: Photo of the FunCube Dongle hardware

Unfortunately, there are drawbacks included with the FunCube that eventually required us to drop it as our receiver.The first was its noise figures. At the center frequency of the beacon transmitter, the FunCube picked up an unacceptable 5dB of noise. This made our GNU radio software struggle at separating noise from signal during transmissions. Another issue we were having was controlling where the FunCube's center frequency actually was. During testing, the frequency of focus would constantly drift during testing, making it almost impossible to maintain a strong signal continuously.

6.3.2 USRP N200

The next system we experimented with the USRP N200 which is a network board developed by Ettus Research. The reason we had given it consideration is because the lab had it on hand from previous projects. This very powerful device had much computational power but it had some major design flaws that proved to be more then problematic.

As mentioned before, the reason we were considering the N200 board is because of its computational power. It included the Xilinx Spartan processor which featured a 100 MS /s dual ADC and Ethernet connectivity capable of streaming data at a Gigabit. The specs on the processor exceeded our needs for the project.

Figure 4.5: Photo of the USRP N200 hardware

This board was almost doomed from the beginning because of one major issue: it did not support frequencies in the 500-megahertz range. This means that in order for our team to even consider use of this board we would require an additional daughter board in order to receive signals in the 500-megahertz range. After the daughter receiver board received the signal, it would be transmitted to the N200, which would then perform computations before transmitting the signal to the pc. This is a complicated process, which requires us to program both boards to function properly. Additionally, the purchase of additional board increases the cost of the system. The N200 costs us approximately $1,720 and the daughter costs about $480, bringing the grand total up to $2,200. Another con of the N200 is its use of Ethernet for streaming data instead of USB. This was inconvenient because it required the installation of different driver on the pc as well as hindering its

upgradability. Fewer computers are including an Ethernet jack in their designs, which the N200 is useless without.

6.3.3 USRP B200

The B200 is another board designed by Ettus Research and is similar to the N200 in terms of processing power with some key differences. It quickly became the clear choice for our design.

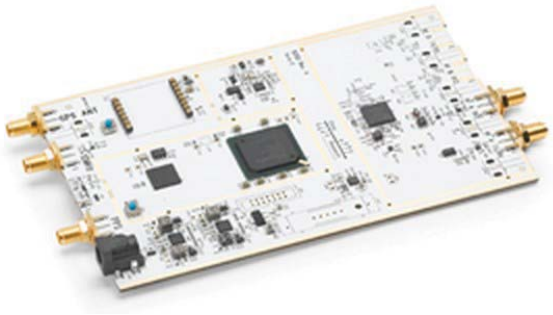


Figure 4.6: Photo of the USRP B200 hardware

Table 6.1: Satellite-defined radio pros & cons chart of front-end hardware solutions

	FunCube Dongle	USRP N200	USRP B200
Pros	· Low Cost · Simple Design · Software Support	· Computing power	· USB data bus · Frequency Range · Computing Power
Cons	· Inaccurate · Large Noise	· Requires two boards · Uses Ethernet bus · Large Cost	· Large Cost · No case

Similar to the N200, the B200 takes advantage of the powerful Spartan processor. A few important differences set the B200 apart, first being its continuous coverage of frequencies up to 6 gigahertz. Instead of using Ethernet for data streaming it made use of the SuperSpeed USB 3.0, which is cutting-edge technology and will be utilized more in

future PC designs. It comes with GNURadio support with the open-source USRP Hardware Driver.

To this point, there have been hard to find many cons to this board other perhaps the cost, which is around $675. This is more expensive than the FunCube but has far better performance. It also lacks a case, which means the device needs to be handled and stored carefully.

6.3.4 Laptop Computer versus Micro Controller

Once we had determined the best solution for our front-end design we had to determine whether we could run the system on a microcontroller such as the Raspberry Pi or if a more powerful processor would be required. Since this SDR is being built with the intent of using it in a mobile setting, size and power consumption are important factors in the design process.

In interest of size and power consumption, the first approach we considered is using a microcontroller, specifically the Raspberry Pi.

Figure 4.7: Photo of the Raspberry pi hardware

The advantages of the Raspberry Pi are its small lightweight design and low power consumption. The only concern we had is the amount of processing it was capable of. Being a Linux machine, it support the GNURadio software that we required but upon further testing we discovered it did not possess the capabilities of performing the computations required by our design.

With computational needs in mind, we concluded that a lightweight laptop with a 3.0 USB is required in order to perform all necessary calculations in real time. Although this design requires more power and is heavier it has its advantages in terms of mobility as well. For a Raspberry Pi, a monitor, mouse and keyboard are all required externally. Comparatively, a laptop has all of these components included in one solid piece, meaning the overall design is far less cluttered and messy.

6.3.5 Doppler Correction Solution Tradeoffs

With the hardware for our system chosen, we now had to consider what approach we would implement to compensate for Doppler shift. Two primary techniques were considered for implementation into our design the first being phase-locked loops.

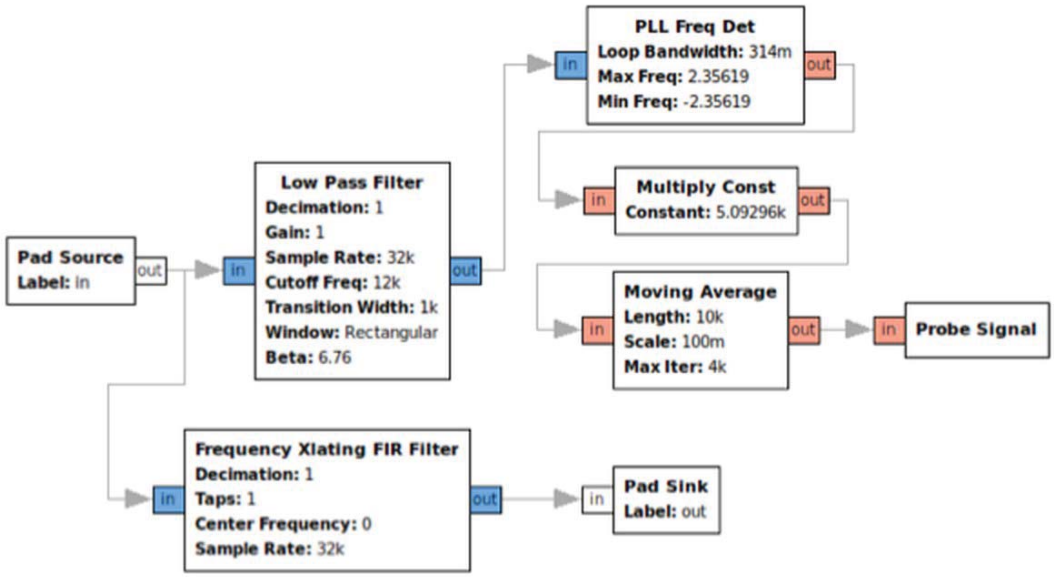


Figure 4.8: Block diagram of phase-locked loop system

What this system does is calculate the phase of the signal received and adjust the phase of the detector to stay in synch with this. In theory, this would be the ideal solution to the Doppler shift issue, but computation was too slow given the rate of change of Doppler or a LEO satellite.

The next approached considered was to predict the frequency of the signal by calculating it via orbital motion equations. Using software called Predict, we took the data about the satellite pass and were able to perform quick calculations of the anticipated frequency. This approached produced more satisfying results and increased the percentage of successfully decoded packets to 85% in tests.

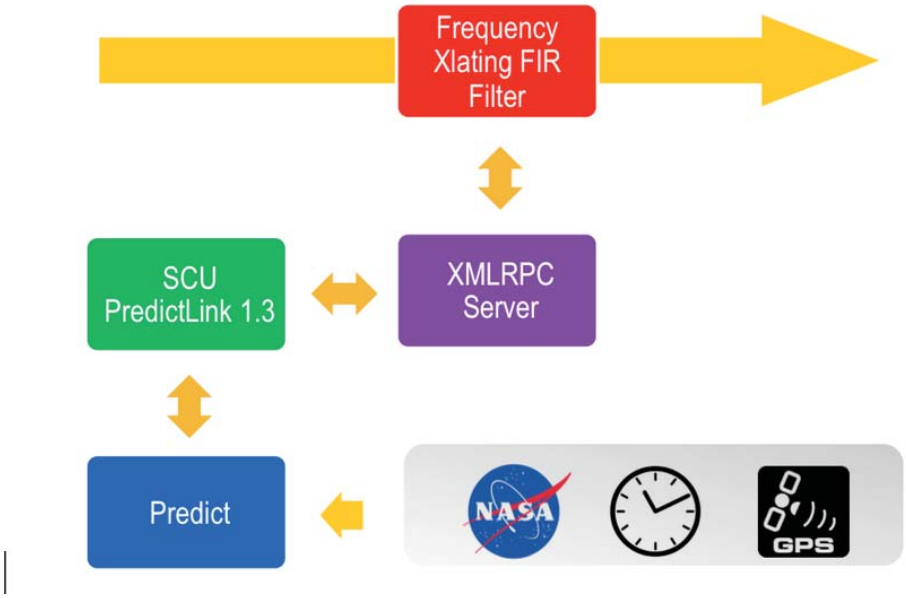


Figure 4.9: Diagram of frequency prediction using Predict software

6.3.6 Consideration of Post-Demodulating Filter Implementation

As previously stated it is recommended to include an additional filter post-demodulation to further reduce noise. In a controlled, ideal environment post-demodulating the signal increased signal integrity and reduced bit flip error by 20%. These results assumed that the system would receive a strong signal with low attenuation while filtering. Unfortunately in the real-life implementation of our filtering system, the required signal strength was seldom reached and inclusion of post-demodulation reduced our ability to successfully decode packets due to high attenuation of the signal. For this reason we decided to exclude the post-demodulator block in our design because signal strength in the field would rarely be ideal.

6.4 Detailed Design Description

6.4.1 Digital Signal Processing

The Digital Signal processing software was designed to be modular, and run on top of the GNURadio processing framework and the UHD driver for intercommunications with the USRP. The system is structured in blocks, which are isolated and independent from each other. Blocks have inputs and outputs, and are connected to each other by media streams. Media streams are unidirectional and convey data in different formats, normally as floating comma complex data, but also as byte streams, real float, etc.

Blocks were developed in C++ when standard implementations did not exist, inter-connection is done in Python, and GUI (graphical User Interface) is based on the standard QT framework. Block parameters that have been code to be modifiable from outside can be modified easily using GNURadio interface or Python language, as well as their inter-connections. More complex modifications and fine-tuning require coding, debugging and compilation. Figure 4.10 shows a simplified diagram showcases the complete DSP system design.

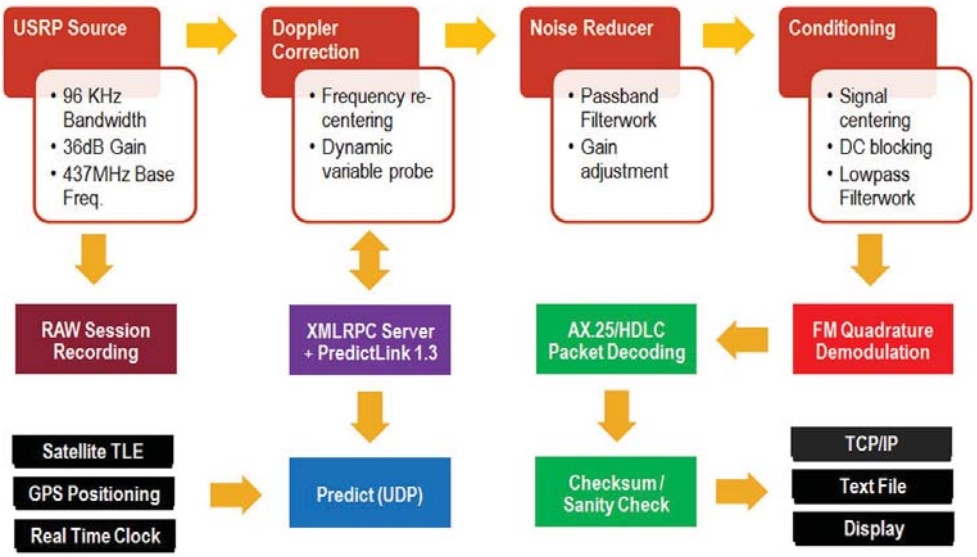


Figure 4.10: Digital signal processing block diagram implementation for the mobile satellite communication station

6.4.2 Signal Sampling and Recording

The initial step is to configure the UHD driver to command the signal input from USRP into the system. Our current, fine-tuned configuration uses the TX/RX Antenna port over USB 2.0 (although USB 3.0 is recommended in supported systems), a gain of 36 dB, and 96 KHz sample rate. The base frequency is set at this point, which depends on the satellite to be tracked. Signal is optionally recorded at this stage into a raw data file using floating comma complex data format.

Alternative sources such as FunCube Dongle Pro are supported and an optimized profile is provided. File playback can be also set at this point, in which case the physical RF source must be deactivated, and a data throttle must be used in order to stream the data out of the file at the desired sample rate.

6.4.3 Doppler Shift Correction

Given the transmit frequency of the OSCAR (Orbiting Satellite Carrying Amateur Radio) system being approximately 437 MHz, we estimated a total Doppler shift of +/- 10 KHz. This value is high enough to require a correction mechanism and take special measures when sampling the digital signal for the first time (Nyquist criteria requires a sampling frequency of at least two times the total bandwidth of the signal).

The design of the Doppler shift subsystem presented some challenges, since it is a serious problem that greatly affects our possibilities for the successful processing of an already weak signal. The initial approach implied using a negative feedback loop analogue to those used in heterodyne FM receivers circuitry in order to lock and track the carrier frequency. After real life testing of the developed solution we reached negative results, and were unable to decode any packet even in laboratory conditions. The final approach involved using the predictable nature of the Doppler effect in order to compute it and feed the corrections to the system.

Solution 1: Implementation of PLL Frequency Detection

The first approach we followed was to use a frequency detector based on a PLL, and then after some mathematical treatment to stabilize the output, use it to correct the offset with a frequency translating FIR filter.

A phase-locked loop or phase lock loop (PLL) is a control system that generates an output signal whose phase is related to the phase of an input signal. Traditionally and in its simplest form, it is an electronic circuit consisting of a variable frequency oscillator and a phase detector. The phase detector compares the phase of the signal from the frequency oscillator with the phase of the input signal and adjusts the oscillator to keep the phases matched. In addition to synchronizing signals, a phase-locked loop can track an input frequency. This frequency detection feature was used in our system to track the peak of the signal from the satellite.

Figure 4.11 showcases how our system was conceived:

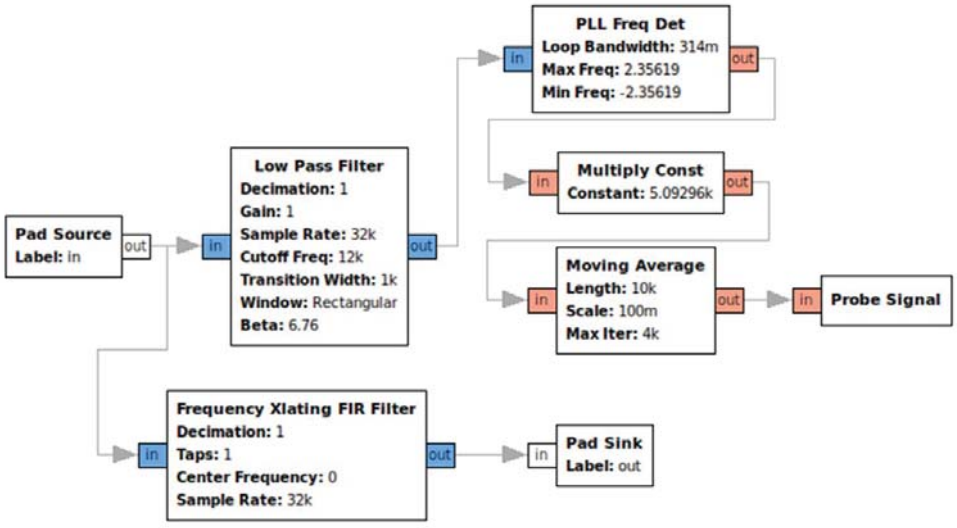


Figure 4.11: Filter data path implemented by the software-defined radio

The signal was taken directly from the source and shifted 20 KHz to avoid the DC peak that some SDR boards like the FunCube Dongle Pro may have (the selected

USRP B200 from Ettus Research does not have it). Firstly, we applied a low-pass filter of 12 KHz cutoff frequency and 1 KHz transition width in order to cut down the signal to the part containing the peak we wished to track. Then it was fed to the PLL Frequency Detector block implemented on the GNURadio Official Tree. Loop Bandwidth was set to 0.314, which in an empirical value that allows the PLL to react quickly enough to the fast changing signal. Maximum frequency is set to 12000($samp_{rate}$/2/3.141593), being $samp_{rate}$ the sampling rate (in this case 96 KHz). 12 KHz accounts for the total Doppler shift of 10 KHz with a small 2 KHz margin, and it is divided into radians per second units as required by the PLL block specifications.

Output of the PLL Frequency Detector contains the frequency of the peak of the signal, which we correct back to Hertzs by multiplying it by a constant with value $\frac{samp_{rate}}{\pi * 2}$. We correct the output with a moving average measurement block configured with a buffer length of 10,000, a rounding scale of 100 Hz and a maximum of 4,000 iterations. This is needed because PLL block output is not very steady when forced to work at such speed. Finally, the signal is fed to a proble block. Another function block is used in the general diagram to read the value and reconfigure in real time a frequency translating FIR filter that corrects the main signal directly after it is inputted into the system.

After several tests we were unable to make this system work effectively, and were unsuccessful decoding laboratory-generated beacons. As real-life conditions would be significantly harder than laboratory ones, we decided it was better to abandon this approach rather than fine-tune the system as it was unlikely that we would succeed in field tests.

Solution 2: Mathematical Doppler Shift Prediction (implemented)

The approach we finally decided to follow takes into account that the Doppler shift can accurately be predicted using a mathematical algorithm that takes into account the Keplerian orbital description of the satellite, the coordinates of the observer and atomic time.

Orbital data of satellites is provided by NASA -among other agencies and bodies-

in a format called TLE (Two Line Element Set). A computer program can use the TLE to compute the position of a satellite at a particular time. A common orbit propagation model is SGP4 (Simplified Perturbations Model 4) which has an error 1 km at epoch and grows at 1-3 km per day(Spaceflight).

Here is an example of a TLE for O/OREOS nanosatellite:

OOREOS

1 37224U 10062C 11076.42207246 +.00000868 +00000-0 +13086-3 001095

2 37224 071.9712 241.3960 0017800 096.5373 263.7815 14.76929834017329

We use an external software named PREDICT, which is designed for UNIX environments, in order to track up to 24 satellite orbits. PREDICT takes the TLEs for the corresponding satellites, a location description file containing the coordinates and altitude of the groundstation, and the system real time clock. Then it generates a normalized Doppler Shift prediction (relative to 100 MHz). PREDICT is able to run in server mode and be polled for this data.

We developed a piece of software named SCU PredictLink (1.3 as current version) using Python. PredictLink is a command line interface that connects via an UDP connection to PREDICT, configures it to the mission parameters, and polls its Doppler Shift data every 50 milliseconds. Then it corrects the normalized frequency to match the tracking frequency of the current mission.

PredictLink is able to interact with the main DSP software via an integrated XML-RPC client. The DSP software runs a server that makes available certain variables to PredictLink or any other client software connected. PredictLink takes current mission configuration from the main flowchart, and pushes back the corrected frequency accounting for Doppler Shift using a TCP link. XML-RPC is a variant of the HTTP protocol that takes minimal resources and allows for almost real time operations.

A frequency translating FIR filter applies the correction at the early stage of the DSP flow. The developed GUI is also able to display real time changes being processed, as well as manual input in case of malfunction.

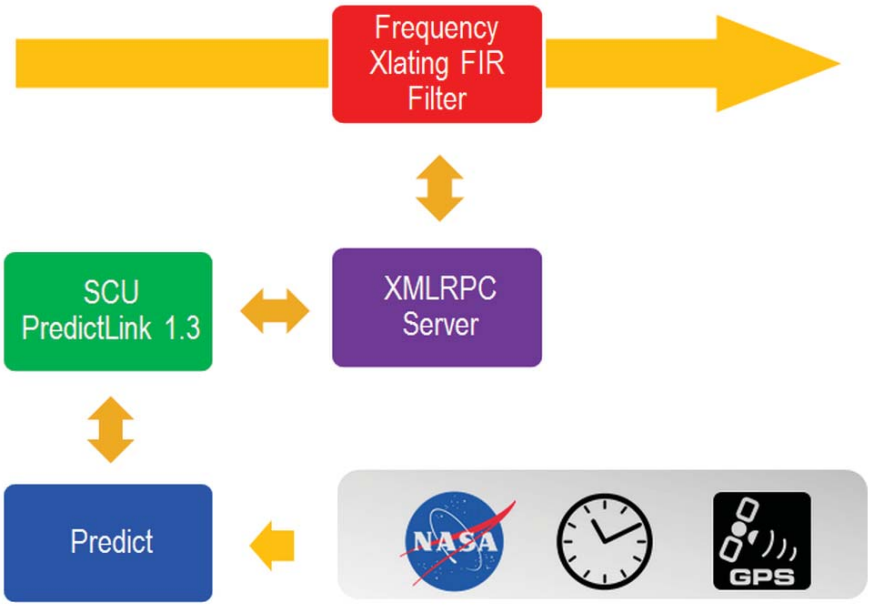


Figure 4.12: Software-defined radio block diagram GUI implementation

6.4.4 Noise Reduction and Signal Conditioning

After Doppler-correcting the signal and having it centered, the signal must be conditioned before being demodulated and decoded. Figure 4.13 shows a clip of the complete diagram shows the main filtering stage of the DSP system.

The first process is general noise levels reduction. For this purpose, a simple power squelch is used. It is configured to be stable and have a threshold of -100dB, meaning every signal or noise under that power will be zeroed-out. Although it does not have direct positive effects in the final system output, it relieves processing power in the following blocks of the flow.

Since the signal normally is not centered at the expected frequency, even Doppler-corrected, a setting must be present that re-centers the frequency and modifies the parameters of all filters to match the new characteristics of the signal. A graphical control is built in the GUI, although it can be preset beforehand. A frequency translating FIR filter is used to move the signal according to the offset in the frequency spectrum.

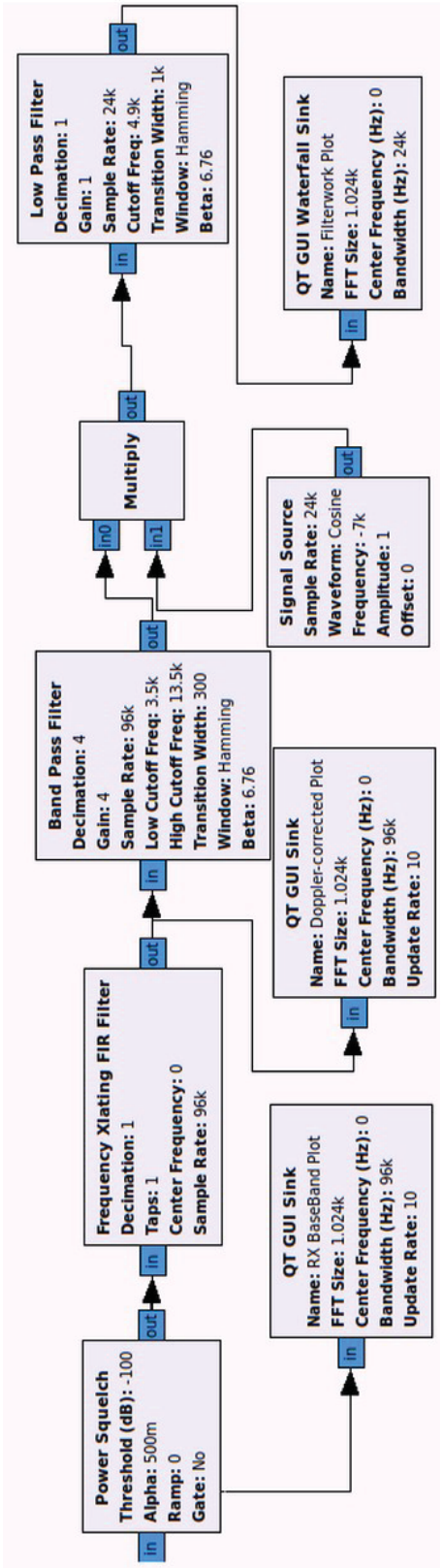


Figure 4.13: DSP filtering implementation used by the software-defined radio

The next step is to perform an initial delimitation of the portion of the input that contains our target signal. For this purpose, a FIR band-pass Filter is used. Its translation width is 300Hz using a Hamming window, and the high cutoff frequency is 10 Khz higher than the lower one. Both frequencies vary according to the configured offset (in 500 Hz increments). The gain of the filter is set to 4, so that output is 4 times stronger than input after filtering. It also implements a rational resampler with a decimation factor of 4. This makes the new sample rate to be 24 Khz. Since the signal has been Doppler-corrected and it is very narrow, using a lower sample rate improves system performance with no signal processing tradeoffs.

The signal is then multiplied by a cosine whose frequency depends on the offset. The result of this is a signal displacement that accounts for the variation in frequency experienced. Signal is now properly centered and ready for the final steps.

Finally, a low-pass filter is applied, with a higher frequency of 4.9 KHz and a transition width of 1 KHz. This is done in order to select the specific portion corresponding to the modulated AX.25 packet transmission.

6.4.5 Demodulation, Post-Filtering and Decoding

Most filter work is done prior to processing the signal using the quadrature demodulation block (which is part of the narrowband FM demodulation needed to decode the signal). However, in certain cases, literature recommends to further refine the output of the demodulator by running it through a filter previous to decoding. We decided to implement a filter and test it to determine its effectiveness. We designed a band-pass filter with low cutoff frequency of 950 Hz and high cutoff frequency of 2450 Hz, considering a narrow transition width of 100 Hz. These were the optimal values to eliminate as much noise as possible from the demodulator output while not attenuating the signal itself.

In our experimental tests in laboratory we found out empirically that under strong signal conditions this filter reduces the rate of bit flip errors by approximately 20%, however when non-optimal conditions were present, many packages could not be decoded. The

reason for this is that the signal was partially masked by the noise, and the AX.25 decoding algorithm seemed to perform better without eliminating such noise. Even though the checksum error rate raised, so did the total amount of decoded packets per minute, which almost doubled. We decided to leave it as an option in the DSP flow, deactivated by default.

Signal must be now separated from its carrier frequency, aiming to recover as much as possible from the original one. As said before, we used the quadrature demodulator block to achieve this purpose, while providing a simultaneous boost in gain.

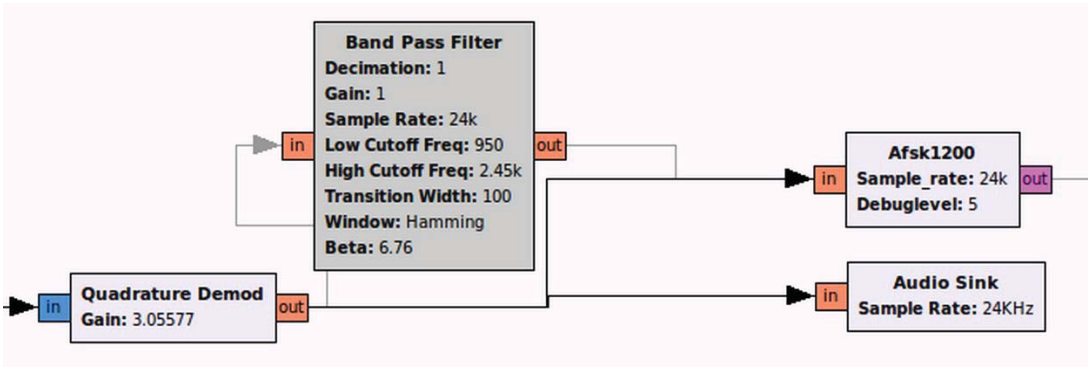


Figure 4.14: Quadrature demodulator implemented in the software-defined radio

Once signal has been demodulated, we use our modified AFSK1200 module to decode the telemetry packages. AFSK1200 is an adaptation of the Multimon software for Linux to GNU Radio, allowing higher sample rates. In integrates a debugger which can be configured to allow different levels of verbosity. Normal operation implies a level of verbosity of 4 or under, which will output successfully decoded packets. Higher levels of verbosity allow for debugging, as well as processing packets that may have a bit flipped and therefore don't match CRC but perhaps are still in good condition to deliver their payload.

6.4.6 Decoded Data Storage and Transmission

Once the AX.25 packets are decoded, the data can be shared and saved as a text file (flat file), uploaded in our web server or uploaded from our iOS Mobile App.

The flat file is a simple .*txt extension file which Unicode 8 is used to save the information gathered from the USRP. In our case, the information received has a timestamp,

the identifier of the beacon (in our case, OOREOS) and an encoded hexadecimal string which contains information about the health of the satellite (i.e power levels, battery level) or the experiment being run in the spacecraft. Our system doesn't parse the data and decrypt the data for security and performance improvement purposes.

Figure 4.15: Example beacon packet from the SporeSat nanosatellite

From an operations point of view, it's important to have the information available in the cloud or at least have it saved on the Internet in case of physical damage of the RV or equipment. We developed a web server capable of uploading the files, retrieving log files and pinning locations using the Google Maps API. The user interface was designed to be easy to grasp and fast to be understood. The root of the page has a HTML5 architecture, and the front-end used CSS and CSS3 for coloring, round corners and design of buttons. In the back-end, we used PHP for authentication, open, read, write, delete and close files on the server and encrypt data. The PHP session variable was used to hold information and pass it to the next pages once the user refreshed the current or loaded a new page. Although it's a useful technique to utilize, the security with such approach is debatable. AJAX (Asynchronous JavaScript and XML) was utilized to load Google Earth API in the message center. The idea of the message center in the web server was to retrieve the hexadecimal string, parse the data and extract the location. Once we had the location, we would input it in the Google Earth API giving us a real time location of the satellites we would be tracking. As a proof of concept, we used the location of the White House with a polygon to include the desired coordinates.

The database schema was based on the MySQL language. The relational architecture has the ability to relate our information in an organized and clear manner. We used tables to store usernames, emails, coordinates of satellites, and timestamps of files uploaded or download. Although MySQL is relatively secure, our website is vulnerable to MySQL injection, which a user can call commands in text fields possibly retrieving usernames and

protected information. The limitation of a maximum number characters in text fields, and the use of a hierarchy approval to modify the database proved to be the best solution in our website.

Lastly, even though the development of an iOS Application was not required, our engineering team decided to create it since it brought facility and fast data sharing when it comes to send the satellite packets to our Santa Clara University's ground station or even final customers. The programmers in the team were more experienced with JavaScript making PhoneGap an ideal solution for our product. PhoneGap is an open source solution for building cross-platform mobile apps with standards-based Web technologies like HTML, JavaScript, CSS. In the front end, we used the JQuery Framework to implement native iOS buttons and visual elements such as the tab bars. In terms of performance, the jQuery framework doesn't present satisfactory performance metrics. Although easy to implement and modify, the iPhone 4S took between 3 to 4 seconds in order to load the user interface on the screen.

One of the functionalities implemented in the mobile app was the ability of pulling data from the web server. This was done creating a XMLHttpRequest object where the server gets updated asynchronously by exchanging small amounts of data with the host. Once the user hits the button to upload the log file, it takes around 1 to 2 seconds to load the content on the screen.

6.5 System Testing and Verification Procedures

The Table 6.2 shows the requirements for the system,and the final results obtained in the latest rounds of testing. In order to test the system and debug it during development, we concluded that using real time satellite passes was largely impractical due to the infrequency of such passes (1-2 per day during working hours) and their length (useful time windows under 8 minutes normally). Therefore we decided to utilize laboratory-grade simulation equipment, which basically consists of the very same communications equipment inside a real nanosatellite contained in a desktop embedded system. In this case we

used the Kittsbox II beacon transmitter configured to mimic the behavior of the O/OREOS satellite at 437302000 Hz.

However this testing was not ideal since the signal strength and noise levels are very different compared to an orbiting satellite, and there is no possibility of effectively simulating a Doppler shift. Therefore once our main system was debugged, we began to use SCU's equipment for tracking a real time satellite. We used our own preamplifier but instead of the 7-element Yagi, we connected it to the rooftop large Yagi antenna whose rotor is controlled by SCU's Mission Operations team and is in active usage condition. We decided to follow this approach in order to be able to test our system before our own rotor tracking system was developed. We also compared our Doppler shift correction values to those being generated in real-time at the Mission Control room by specialized equipment.

Table 6.2: Chart showing the SDR requirements and achievements

Requirements	Achievements
Friendly user-interface for operators	Fully developed GUI for main controls
Implement Doppler Shift Compensation	Mathematical real-time orbital prediction
Flexible and Portable Design	Installed successfully onboard vehicle
Maximum Set up time: 10 Minutes	Average set-up time: 4 mins
Rate of Decoded Packets: 80 %	85% Success (Lab-simulated Conditions)
Ability to Record Passes	Feature Included (Raw & Doppler-corrected)
Multiple Satellite Accommodation	24 Simultaneous Satellites
Retrieve Decoded Packets	File, TCP / IP, Web Server, iOS Application
Cost Reduction (Existent: $2,000)	$974 (54.9 % Reduction)

6.5.1 Phase 1: SDR Equipment Pre-testing

Prior to going through our whole-system tests, we performed a series of proof-of-concept experiments in order to verify that our chosen hardware set-up was properly working and that it matched our expected performance parameters.

In order to do this we developed a simple Stereo FM Radio receiver, a NOAA

Weather Camera receiving system, and a Mode-S Secondary Radar telemetry receiver from aircrafts taking off and landing at the nearby San Jose Mineta International Airport (SJC). These three tests permitted us to understand the inner workings of an SDR-based DSP as well as verifying three different environments: a fixed-point transmission reception (FM Radio), an orbiting object data reception (NOAA) and a non-orbiting object moving object data reception (Aircraft).

6.5.2 Phase 2: Beacon Reception Testing with Simulator

For the development of the real SDR subsystem described in this chapter, we used the Kittsbox II NASA Nanosatellite Beacon Transmit Simulator. In our setup, The Kittsbox was connected to a moderately low-gain omnidirectional antenna and was placed 20 meters away from the laboratory where the tests were conducted (with a room of separation in between and on the same floor). We tuned it to 437302000 Hz and used the O/OREOS telemetry beacon pattern transmitted every 5 seconds. We also used temporarily the Sporesat pattern for a limited time in order to avoid interference with a running mission, but to this respect the testing conditions did not vary.

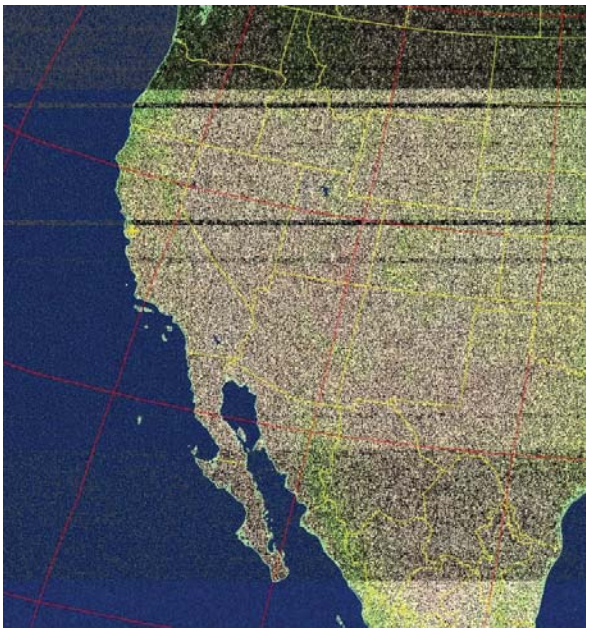


Figure 4.16: NOAA image pulled from SDR system

In the initial stages a complete set-up of the USRP, the pre-amplifier (fed with 12V CC power supply) and the Yagi pointed manually towards the source were used. In later stages, once the raw data recording feature was implemented, there was no further need to use a real setup since the beacons could be recorded and re-played ”offline” as many times as it was needed.

In our tests we were able to decode approximately 85% of the packets received on average, meaning out of 100 beacon packet transmissions 15 were discarded because of a bad checksum or were simply not decoded successfully.

For debugging purposes our decoder implemented a high-verbosity setting which outputted raw information even if CRC checksums were incorrect. This allowed us to manually interpret some of the data and eliminate false negatives.

6.5.3 Phase 3: Beacon Reception from O/OREOS Orbiting Satellite

Once the system was optimized and proven to work under laboratory conditions, we modified our tests in order to reflect real life conditions. We chose to monitor the O/OREOS satellite passes because of two reasons:

1) It was the same onboard communications systems that we used for our laboratory tests with the Kittsbox, therefore we reduced the chances of a premature incompatibility of our design.

2) Its passes were frequent enough and beacon data was transmitted every 5 seconds which is an optimal value. Other nanosatellites transmitted every few minutes, making it much harder to test the system.

It took several passes in order to adjust the system. There were two objectives: first, test if the Doppler shift correction was working; second, to successfully receive a real packet.

Doppler shift correction proven to be more tricky than we expected. Several trial-and-error iterations were required to fine-tune the system, which originally had positive feedback instead of negative because of a design flaw, and did not sync correctly to the

real time clock. After these corrections we verified that our Doppler shift prediction was on average +/- 60 Hz off from the one being generated by the Mission Control equipment. This result was outstanding since it is a very low error margin compared to the total signal bandwidth.

We faced several issues for decoding live packets. The main one was the very poor SNR (Signal to Noise Ratio), as the amplifier was not powerful enough to boost the signal to expected levels. Also, we had several interferences from unknown sources, and the issues faced with Doppler shift correction made it impossible for the system to work until it was fully operational and signal was being corrected. Although the software performs the functionality it was designed for, it didn't account for external factors such as interferences of outside sources such as Wi-fi signals or cellular data bands. Later, the software has to be upgraded with infinite impulse response (IIR) bandpass filters to attenuate the effect of such frequencies and maximize the band of the satellite's center frequency.

After several tests we made the first partial packet decode on May 2nd, which is showcased below:

```
Packet: ?????@???l???‘?????@a ?OOREOS.org 88Ew
??[Payload: OOREOS.org 88Ew??
[Payload(hex):[, 4f, 4f, 52, 45, 4f, 53, 2e, 6f, 72, 67,
20, 20, 20, 38, 38, 45, 77, be, 3f, 5b]
PID: f0
CTRL: 3
SRC: KF6JBP
DEST: UNDEF
```

6.5.4 Current Development Status and Pending Work

At present time, the whole SDR subsystem has been developed and tested under laboratory conditions, at least to the extent required to meet the minimum requisites set for the project. In terms of hardware, the entire PC-preamplifier-antenna and its cabling and power needs have been addressed. Presently the setup consists of a desktop computer running Ubuntu Linux 12.04 LTS, basic preamplification for the specific band required, and a

generic 7-element Yagi antenna. In terms of software, the whole DSP design is finalized in an operational status under lab conditions (based on GNURadio), and the Doppler Prediction system is also developed. Parallel developments such as a basic web server and iOS app are in ”beta” status and can be considered functional.

However, there are some aspects that were not finalized for time or technology knowledge constraints, which will be needed for an optimal operation under real-life conditions. They are as follows:

- The current packet decoder module (AFSK1200) works only with a reasonably high SNR (Signal-to-Noise Ratio). It may need to be fine tuned in order to increase packet decoding rates in non-optimal conditions where the signal strength is low and there is a noisy environment. The C++ source code of the module will be made public in order to enhance possibilities of future development.

- Presently, SNR is very poor with real-life satellite transmissions. Further research and experimentation with other preamplifiers may be required in order to boost the signal and increase the chances of decoding a packet.

- While packets have been successfully decoded with a rate over 85% under lab conditions using the 7-element Yagi, the system has only be tested with the larger Yagi installed at SCU ground station. Even with a better antenna, results were poor, although the longer cabling may have contributed to this situation.

- The USRP Board will require a proper casing in order to protect it. Also, the current desktop PC will need to be replaced by a laptop for increased mobility and reduced power consumption.

- Some of the software developed, like PredictLink, requires command-line launch. A more integrated interface and user experience will facilitate the work of the operator and save a significant amount of setup time.

Chapter 7

Auto-Calibration

7.1 Overview

In the context of a mobile satellite communication station, calibrating the dish antenna to reflect the orientation and location of the vehicle is an important part of correctly tracking the satellite. The mobile vehicle's GPS location, altitude, yaw, pitch, and roll with respect to the surface of the earth will need to be accounted for differently than a static station, such as the one used by the RSL, where those offsets can be compensated for initially with hardware.

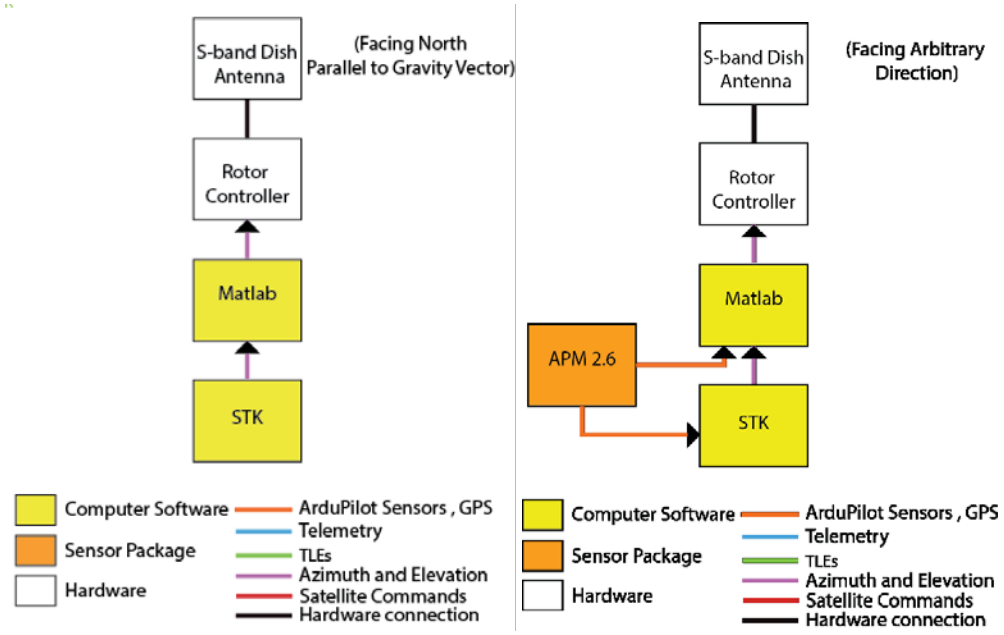


Figure 5.1: RSL tracking block diagram flow shown on the left compared to the mobile ground station tracking block diagram shown on the right including auto-calibration

The difference in operations can be seen in Figure 5.1 where the RSL tracking block diagram operations are shown on the left and the mobile ground station tracking block diagram operations are shown on the right.

By finding these values dynamically, the station can compensate quickly for errors in pointing caused by the vehicle's orientation, location and altitude. A degree of pointing error has a noticeable negative impact on the dish performance. In order to operate with maximum performance, the auto-calibration system is required to accurately point to the satellite at all moments of contact with less than a degree of error.

7.2 System Options and Tradeoffs

The alternative to not using an auto-calibration system is that a manual calibration would need to be done every time that the station was set up. This would involve carefully aligning and leveling the trailer and then checking to see if the dish is properly tracking the sun. This method involves using the shadow cast by the antenna feed and adjusting the dish offsets so that the shadow of the antenna feed is centered in the middle of the dish. This verifies that the dish is pointing directly at the sun assuming that the antenna feed is centered properly on the dish. This could potentially be a difficult task when considering a mobile station. The dish is located above the top of the trailer and it is difficult to see the interior face of the dish. Also if a pass is being conducted in a shadowy region or at night, there is no method for calibration. Implementing such an auto-calibration system required new hardware and software to be developed whereas the manual calibration method could be done at no monetary cost.

Another alternative is purchasing individual components to determine orientation and location. This solution requires more development in the sensor hardware in order to make it compatible with the current architecture used by the RSL. A benefit of different sensors is that they may be more accurate and less noisy compared to the APM 2.6. The APM 2.6 was ultimately chosen due to its well established design, documentation, and existing APIs. APM software already exists allowing for the sensors to be visible in the

console window allowing for very little setup process of the APM. The APM is normally sold as a highly capable but low cost autopilot for unmanned aircraft. Because it includes all sensing needs in a low-cost and easy-to-use package, it was adopted for use here.

7.3 Design Description

The auto-calibration system makes use of three magnetometers that are implemented in an APM 2.6 board shown in Figure 5.2. A GPS is also seen in Figure 5.2 and is hooked up to the APM 2.6 in order to provide GPS location while the magnetometers relay information about the yaw, pitch, and roll of the vehicle with respect to the surface of the earth. The APM is located on the mast of the S-band dish to obtain the most accurate measurements for calibrating the antenna.

Figure 5.2: Photo of APM 2.6 Sensor Package

The yaw, pitch, roll, altitude, latitude and longitude is fed to the main analysis computer through an RF connection. This information is then displayed in a console window and sent to MATLAB for analysis. The GPS information is stored in the database to be used by Satellite Tool Kit (STK). STK uses the new GPS coordinates to generate satellite

specific visibility times, creating a list of azimuth and elevation values for the satellite for every second of visibility which is stored in the database for later access. These original azimuth and elevation values assume that the dish antenna is located on level ground and is aligned north. MATLAB then uses the yaw, pitch, and roll values from the APM 2.6 and builds a rotation matrix that converts pointing angles in the flat ground, pointing-north frame to angles in the frame of the antenna. The rotation matrix generation will be described in more detail in the next section. MATLAB then pulls the pass information list generated by STK from the database and applies this rotation matrix to each azimuth and elevation pair which results in a new azimuth and elevation trajectory with respect to the frame of the vehicle. During a pass, the tracking software accesses these values from the database and send the commands to the rotor controller, which controls the dish rotor to track the satellite.

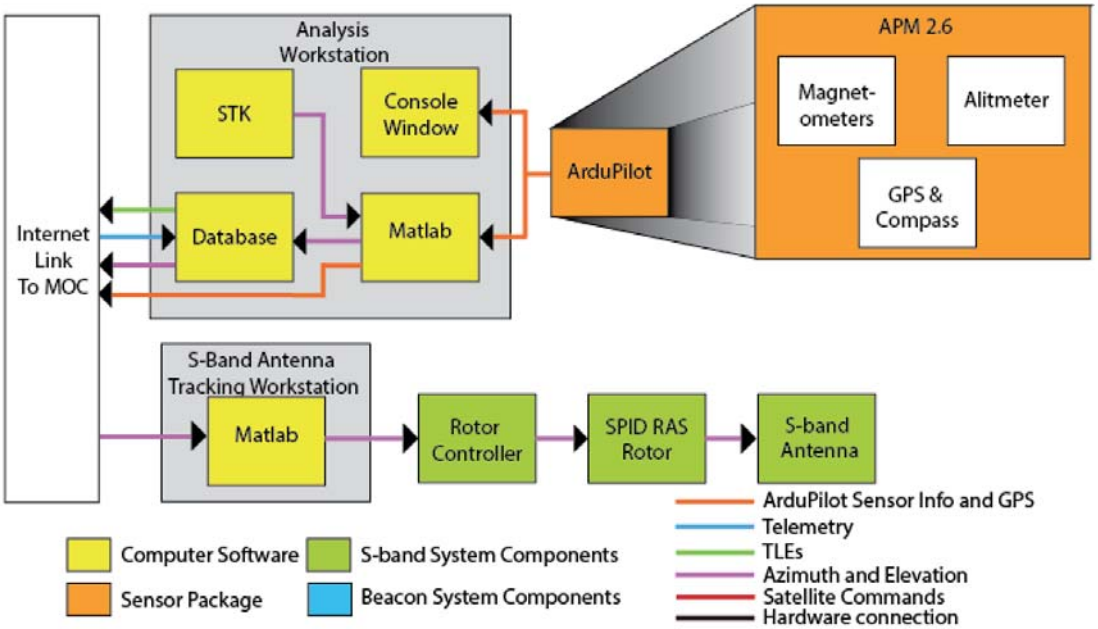


Figure 5.3: Auto-calibration system block diagram used in the mobile ground station

The system does not create pointing information dynamically. The values are stored and accessed later due to the settling time of the sensors. If a dynamic solution is to be implemented, more accurate and faster sampling sensors should be used. Figure

5.3 shows the software components and the process of operations.

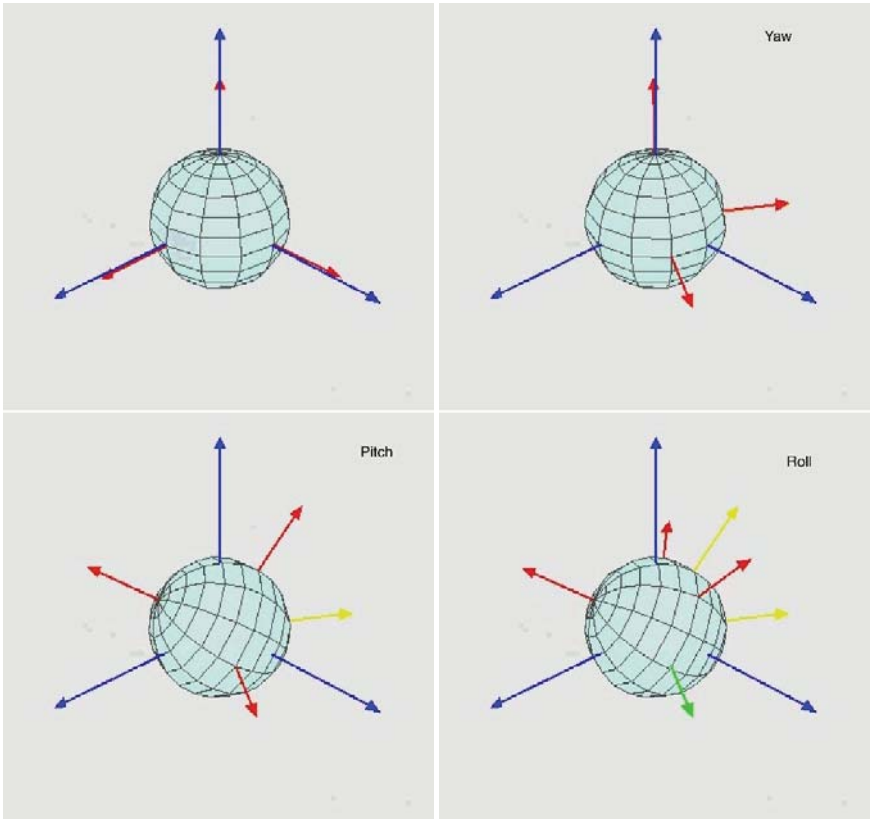


Figure 5.4: Physical steps of frame rotation compensating for yaw, pitch, and roll offsets

A physical representation of this process can be seen in Figure 5.4 where the sphere 3 coordinate axis represents the frame of the trailer. The initial position in the top left is the frame that STK assumes when generating azimuth and elation information. The bottom right corner shows an example corrected frame. The system does not experience three rotations before it is pointing correctly, but the rotation effectively takes all these corrections and applies them in a single operation.

Consider the following offsets of yaw, pitch and roll denoted by α, β, and γ respectively. First the yaw is taken out through an α degree rotation about the z axis. Then pitch is compensated for by β degrees of rotation about the y axis. Lastly a γ degree rotation about the z axis completes the frame conversion. The diagram of this transformation can be seen in Figure 5.5.

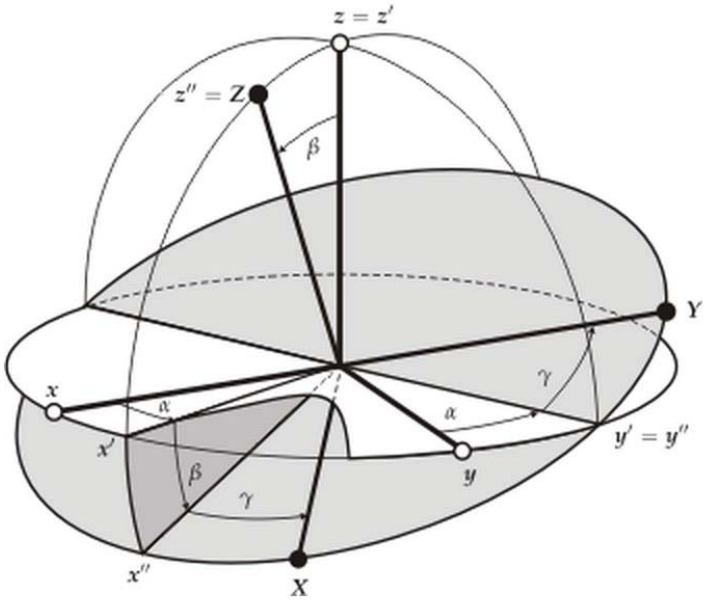


Figure 5.5: Frame transformation doing a Z, Y', then Z" rotation about α, β, and γ angles

7.3.1 Rotation Matrix

The rotation matrix used for this operation is written in Equation 5.1. Since the sensors give the yaw, pitch and roll with respect to the global frame (level ground and pointing north), an extrinsic rotation can be done. An extrinsic rotation consists of three separate rotations about a fixed axis. The matrix written in Equation 5.1 is an expression of three intrinsic rotations about the z axis, y axis, then the x axis by offsets of yaw, pitch and roll denoted by α, β, and γ respectively.

$$R = \begin{pmatrix} cos(\alpha)cos(\beta) & cos(\alpha)sin(\beta)sin(\gamma) & sin(\alpha)sin(\beta) + cos(\alpha)cos(\gamma)sin(\beta) \\ cos(\beta)sin(\alpha) & cos(\alpha)cos(\gamma) + sin(\alpha)sin(\beta)sin(\gamma) & cos(\gamma)sin(\alpha)sin(\beta) + cos(\alpha)sin(\gamma) \\ -sin(\beta) & cos(\beta)sin(\gamma) & cos(\beta)cos(\gamma) \end{pmatrix} \quad (5.1)$$

7.4 Supporting Analysis

The auto-calibration system was implemented with both the G-5500 rotor and the SPID RAS rotor. The APM 2.6 was attached to the top of the rotor during this stage of prototyping and plugged directly into the analysis machine with a USB cable. The rotors

were given a known tilt and the yaw, pitch and roll of the rotors were recorded. Satellite Tool Kit generated azimuth and elevation information for the O/OREOS satellite and stored them in a database on the analysis machine. MATLAB took the sensor information from the APM 2.6 and generated new azimuth and elevation values. This process was done for every combination of yaw, pitch and roll - e.g. yaw only, yaw and pitch only, etc. The tilted rotor then tracked against a rotor on level ground that pointed north to verify that the rotors pointed the same direction.

Results showed tracking was identical between both rotors. During parts of the pass the rotors were a degree off from each other for a second at most but this can be attributed to the deadband threshold settings on the rotor controllers. Regardless each rotor still maintained less than one degree of error tracking.

7.5 Test and Verification

Table 7.1: Comparison of azimuth and elevation values corrected by auto-calibration with 7 degree yaw offset

$Azimuth_{stk}$	$Azimuth_R$	$Elevation_{stk}$	$Elevation_R$
303.809	310.809	11.907	11.907
303.671	310.671	11.990	11.990
303.532	310.532	12.074	12.074
303.392	310.392	12.157	12.157
303.251	310.251	12.241	12.241
303.109	310.109	12.325	12.325
302.966	309.966	12.409	12.409
302.822	309.822	12.493	12.493
302.678	309.678	12.578	12.578
302.532	309.532	12.662	12.662

The auto-calibration system was tested in two ways - comparing the data that was generated by using the software and optically comparing rotor tracking between a rotor using the auto-calibration software and a rotor not using the software. It was difficult to determine the correctness of the software by looking at the data being generated. Only in the case of yaw only offset was it clear that the software was operating properly. Several azimuth and elevation combinations can be seen in Table 7.1 where the STK generated

azimuth and elevations are compared against the auto-calibrated azimuth and elevation values considering a yaw offset of 7 degrees. Due to the other combinations of offsets that will exist in practice, verifying rotor pointing with a known working system was needed.

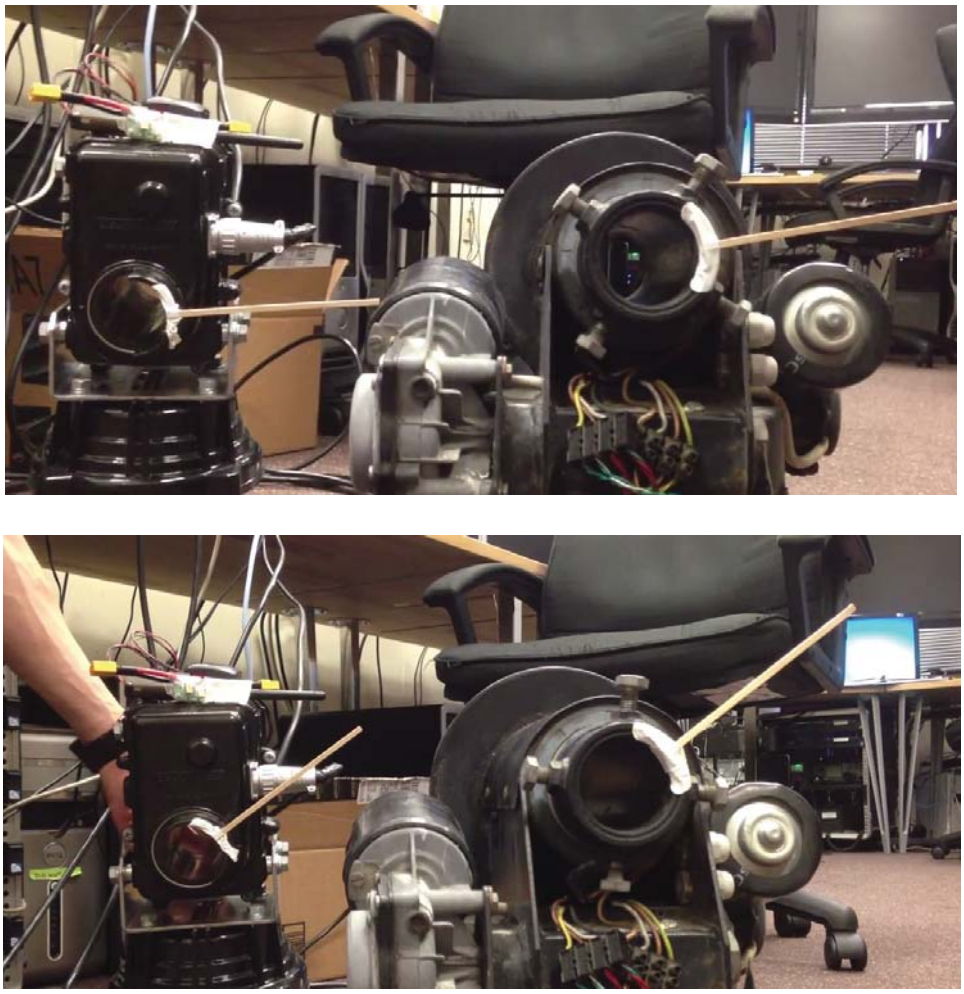


Figure 5.6: SPID RAS rotor tracking on uneven ground next to G-5500 rotor tracking on even ground

The Robotic Systems Laboratory uses a piece of software called NOVA to send tracking commands to the rotor controller. This software assumes flat ground and that the antenna is pointing north. A G-5500 rotor was placed on level ground and faced north. The G-5500 was then connected to a computer running NOVA and tracked the O/OREOS satellite. Simultaneously a SPID RAS rotor was oriented in a way so that it was experiencing yaw, pitch, and roll. Then the auto-calibration package fed information to the SPID RAS

rotor controller to point at the satellite compensating for the new frame. The rotors were placed close enough that a significant difference in tracking would be seen. Two pictures of the setup can be seen in Figure 5.6 where two rotors are seen tracking the same object using wooden sticks to indicate the pointing direction.

At parts of the pass the rotors were a degree off from one another. This is due to the deadband threshold settings on the rotor controller. This setting determines the resolution of azimuth and elevation, that is it will only move to the next degree and will not recognize half degree commands. This setting is kept because the rotors experience substantial vibration when moving. The number of movements needs to be minimal while maintaining the least amount of error possible. One degree of error allows for good enough performance for satellite communication while minimizing rotor stresses from vibration.

Chapter 8

System Integration and Testing

8.1 Overview

The subsystems developed for the mobile communication station have high modularity. The S-band system is able to operate the spacecraft independently from the software-defined radio. Conversely the software-defined radio can run on its own without S-band communication. This allows for the subsystems of the station to operate even when one is in a mode of failure. The auto-calibration is not vital toward operational success either. The trailer can find level ground and operate from that location while ensuring the dish points north. Although failure of subsystems is not desired, a single subsystem failure does not necessarily jeopardize the entire operation. Although the subsystems do not rely heavily on each other, they are connected through computer workstations.

The auto-calibration is accessed by both the S-band and software-defined radio systems during nominal operations. As seen in Figure 6.1 the sensor package used for auto-calibration is connected directly to the analysis workstation. The S-Band tracking workstation and the SDR workstation both access the database on the analysis workstation for the azimuth, elevation, and range (aer) information for tracking. The SDR also stores data onto the analysis workstation for further processing. From the Figure 6.1 it is also evident that integration is mostly through the workstations which will be discussed next.

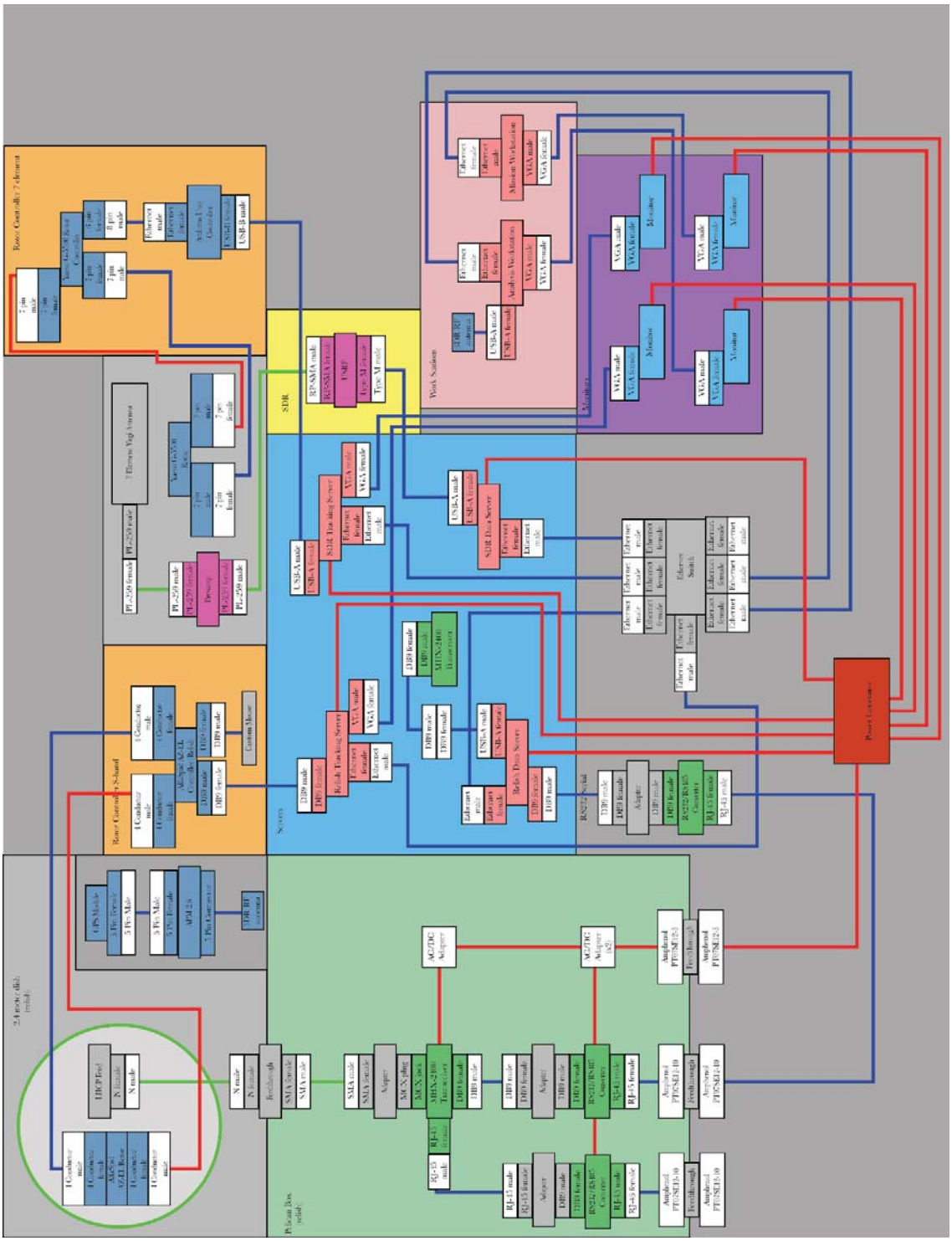


Figure 6.1: Complete Component Block Diagram of Mobile Communication Station

8.2 Workstation Integration

The integration of the workstations involved hard cable connections denoted by the complete block diagram displayed in Figure 6.1. The analysis workstation's main connections include the tracking workstations as well as the SDR data workstation. Although the commands for tracking are generated by the antenna specific workstations, the pointing information is generated by the software on the analysis workstation and the APM 2.6 sensor information. The data from the SDR data station is processed in MATLAB running on the analysis station as seen in Figure 6.2.

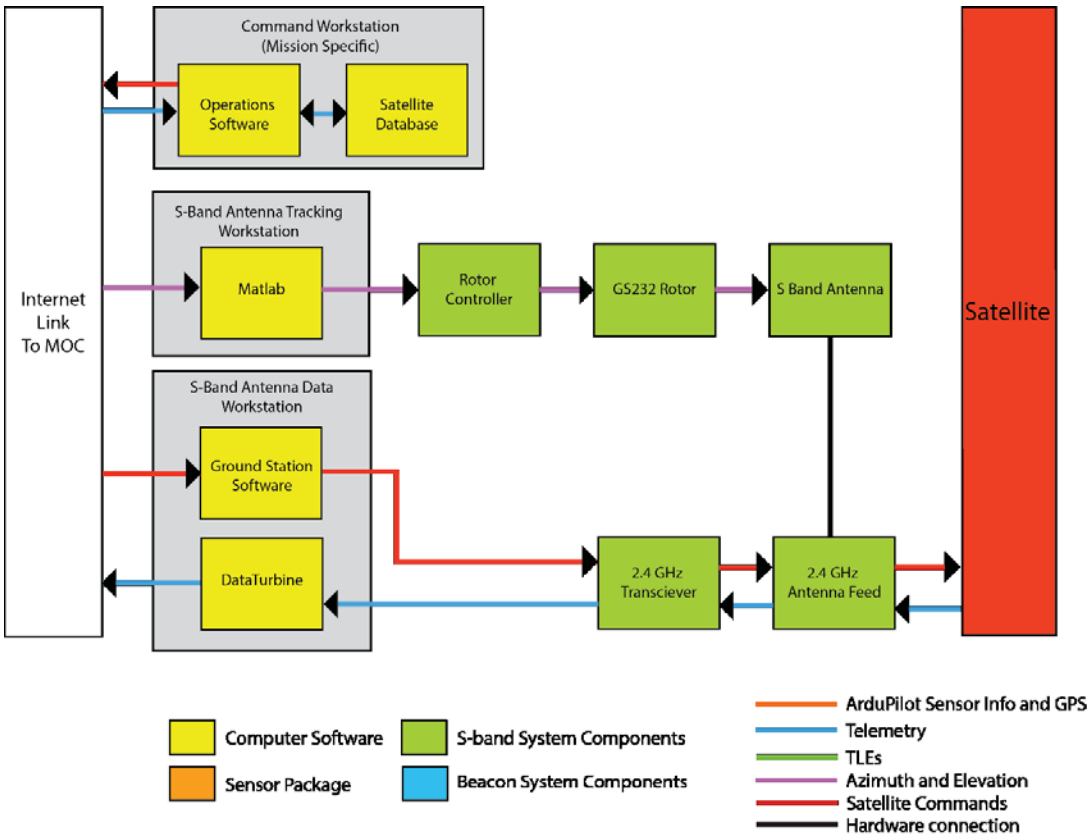


Figure 6.2: SDR Component Block Diagram of Mobile Communication Station

The command workstation is connected to the the S-band data computer which is responsible for sending commands to the MicroHard radio connected to the dish antenna. The MicroHard then receives data from the satellite which is transferred back to the com-

mand workstation for processing. The data flow for S-band communication can be seen in Figure 6.3.

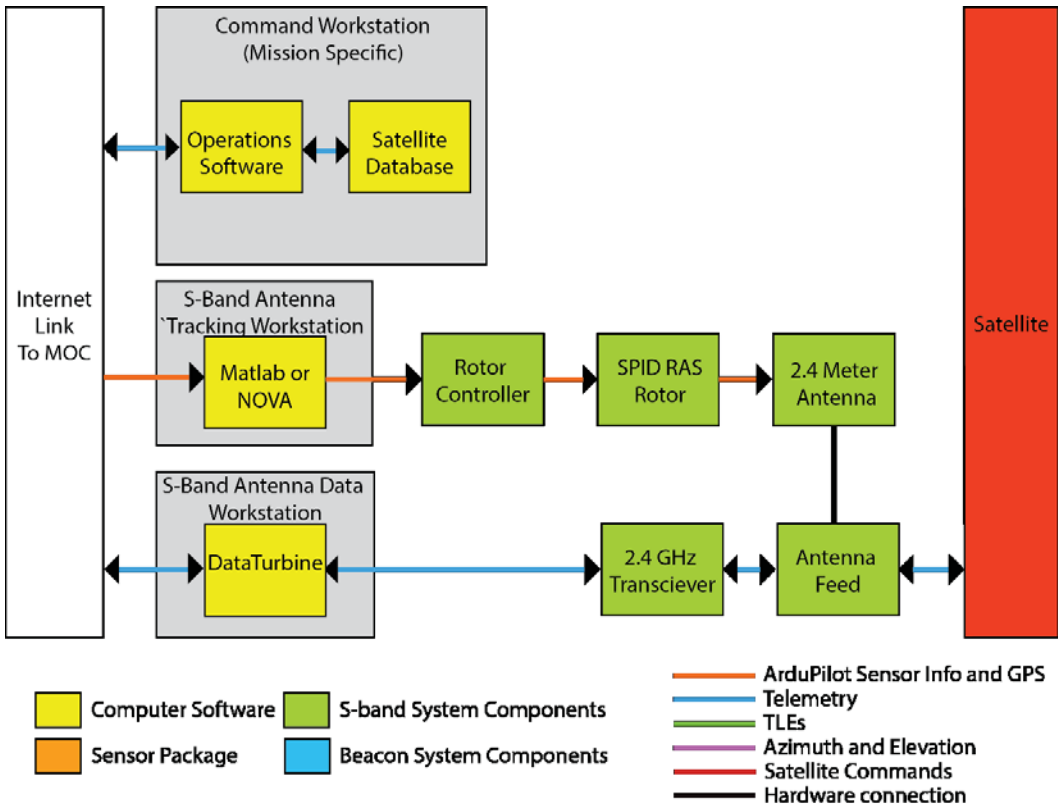


Figure 6.3: S-band Component Block Diagram of Mobile Communication Station

8.3 Experimental Protocol

The experimental protocol for our system determines whether the design requirements were met for our system. The functional requirements are tested first to ensure that the technical specifications matched the expectations of the customer - the RSL.

The most important feature that the system bring to the RSL is the increased pass availability. Unfortunately, testing this requires that the most complicated subsystem, the S-band antenna, is working properly. Since that subsystem is not complete, this testing will be performed by a future team of students. The system is tested by running O/OREOS satellite contacts and recording the amount of data downloaded during a pass. The trajectory of the pass as well as maximum elevation is recorded in order to compare the amount of

data downloaded compared to a similar pass that has been run in the past. The metric used is percentage of data downloaded compared to similar O/OREOS pass data download quantities.

The second most important system on the trailer is the software-defined radio because it too is responsible to handling communication with the satellite. This is a much easier test to run because it only requires that the SDR system is working. The software-defined radio is tested by tracking the O/OREOS satellite and comparing the performance to the existing beacon stations used by the RSL. The metric used will be number of beacon packets decoded compared to the existing beacon stations.

The least critical system on the mobile communication station is the auto-calibration since testing can be done on even ground. However, auto-calibration integration is a more difficult test because the only metric to determine the accuracy of the pointing during track is the amount of successful commands that go through to the satellite. Measuring the actual angles that the antenna is pointing at is nearly impossible and the team does not have the ability to perform tests of this caliber. The closest way to test is to park the trailer near a static antenna on uneven ground and visually inspect the accuracy of pointing.

8.4 Testing and Results

Due to time constraints, the team was not able to execute the experimental protocol. Individual subsystem testing was performed; however, the lack of subsystem integration with the mobile vehicle did not allow for nanosatellite operation testing. There were several requirements we were not able to test successfully. Since we were not able to perform S-band communication with a live satellite, the increased amount of communication availability and possible link margin could only be inspected through link budget analysis. The stowed volume of the S-band system was tested and proved to be less than the 1.5m x 1.5m x .7m volume. The next step for this subsystem is to integrate it with the vehicle and operate the O/OREOS satellite and ensure that the performance is close to what was expected.

The software defined radio beacon system was not able to successfully receive the

80% of decoded packets that was stated in the requirements. Although further debugging is needed, the next step for the SDR is to integrate it with the mobile vehicle and begin using the Yaesu rotor for tracking. Despite not being able to decode packets, the SDR did reduce costs compared to the current RSL beacon system solution. With the other subsystems integrated the auto-calibration system can be tested as well. Although a more difficult system to verify, ensuring that the communication performance meets the expected data rates will be a sufficient test.

Overall the team was able to estimate a setup and breakdown time of 24 hours by timing the assembly and disassembly of the major components such as the dish antenna, feed support mechanism, antenna mast, and base support structure. Though the estimation suggests this requirement was met, further testing and verification will be done by a future team of students involving a complete setup and breakdown of all subsystems. The team met the collapsable volume constraint of less than $1.5x1.5x.7m^3$. Also individual subsystem testing and analysis verified working subsystems however testing with live satellites was not performed. The next steps for this project is to integrate the subsystems with the mobile vehicle and run passes with the O/OREOS satellite. After testing it will be clearer what parts of the design need to be changed for the system to run optimally. It will also determine if the subsystems are capable of operating a nanosatellite using the 2.4 meter dish. It is possible that a three meter dish will be required to operate successfully.

Chapter 9

Costing Analysis

9.1 Budget and Costs

Table 9.1: Complete Cost Report for Mobile Communication Station

	Budget Update				
TEAM	Mobile Satellite Communication Station				
Date	5/8/2014				
INCOME					
Category	Source	Sought	Committed	Pending	
Grant	Undergraduate Programs Senior Design Project Fund	$1500	$1500	$0	
	SCU Technology Steering Committee	$10000	$10000	$0	
	The Robotic Systems Laboratory	10000	$10000	$0	
	TOTAL	$21500	$21500	$0	$21500
EXPENSES					
Category	Description	Estimated	Spent	Pending	
Operations Vehicle	Rec. Vehicle	$10000	10000	$0	
Independent Workstations	Computers	$900	$900	$0	
	Cabling	$200	$200	$0	
S-Band System	Tracking Computer	$600	$600	$0	
	Data Computer	$600	$600	$0	
	2.4 Meter Dish Antenna	$1396	1396	$0	
	Antenna Feed	$165.71	$165.71	$0	
	MHX radio	$795	$795	$0	
	SPID RAS Rotor and Controller	$2129.57	2129.57	$0	
	Support Structure	$257	$257	$0	
	Misc Hardware	$325.6	$325.6	$0	
Software-Defined Radio	USRP B200	$675	$675	$0	
	7-element Yagi Antenna	$60	$60	$0	
	LNA Preamp (4420B)	$127	$127	$0	
	Cables and Connectors	$40	$40	$0	
	Data Computer	$600	$600	$0	
	Tracking Computer	$600	$600	$0	
	G-5500 Rotor and controller	$749.95	$749.95	$0	
	Arduino Uno	$24.71	$24.71	$0	
Auto-Calibration System	APM 2.6	$129.88	$129.88	$0	
	TOTAL	$20375.42	$20375.42		0
	Net Reserve (Deficit)		$1124.58	$0	$0

The entire mobile ground communication system cost was $20,375.42 as can be seen in Table 9.1. This total system cost was below our $21,500 budget for the project; however, this budget was used for any new hardware that needed to be purchased to complete the project. Many of the expensive hardware components were already in the RSL's possession in anticipation of constructing a new ground station. Some of these included the rotors, workstations and support structure, so the amount of money spent for new hardware was a little over half of the budget.

Reducing the cost of the software-defined radio system was the only monetary constraint we had through our design. Since the station will be used for future satellite missions, and maintenance cost is low, having costs similar to the the price of building a static station was sufficient for this project. The software-defined radio did succeed in reducing the beacon subsystem cost as discussed in the previous SDR section.

Chapter 10

Business Plan

10.1 Executive Summary

The mobile ground station is a satellite communication station designed to talk to nanosatellites. It has the ability to command satellites that use the S-band frequency as well as the ability to listen to satellites that use amateur beacon frequency. The mobile ground station can be contracted to operate any satellite missions that use these two methods of communication. Nanosatellites are the main target market for this communication since missions are much shorter than larger scale satellite missions. A good number of nanosatellites are experimental and quick data download becomes an attractive feature when missions have a short duration. This station will offer increased data download from other static stations using the same technology.

10.2 Background

The goal of the mobile ground station is to help increase satellite communication efforts by allow more communication with the satellite. The mobile ground station has the mobility advantage over many satellite communication stations. This allows it the ability to take advantage of the satellite's orbit and operate at the optimal location for the most frequent satellite visibility. The operation of nanosatellites are the target market and these satellite generally have short lifetimes. Due to the nature of these satellites our business model is to contract out the mobile satellite communication station for the duration of the mission.

The current market for operating nanosatellites is increasing and can be seen through the number of satellite missions being contracted to the RSL. In 2013 the RSL had one nanosatellite launch that they now operate. In 2014 however, the RSL have had two satellites launched already and plan to have a few more before the year is over. Companies interested in renting the mobile communication can employ members of the RSL to run the system rather than teach members of their team how to use the complex system.

The competition that this product faces is other communication stations that operate nanosatellites. However there are almost no commercial mobile nanosatellite communications that we are currently competing against. Mobility is also a large attraction of this product, allowing the customer more flexible operations. Operating from a static facility, they may not be able to collect as much data as a mobile application.

10.3 Key Technologies and Customers

The technologies that the mobile ground station offers is S-band communication and amateur beacon frequency reception at a location in the United States of their choosing. This allows a great deal of flexibility for a nanosatellite team, allowing them to choose a location near their operations facility or in a location that best suit the satellite's orbit. The station also comes with all the hardware and software necessary to run successful satellite missions.

NASA is a likely customer for our system. With several launches planned for 2014 and 2015, they will likely be in need of another control node. Furthermore the biology experiments that they plan to launch require a certain amount of data to be downloaded from the satellite to reach minimum success. By offering another station that can move to an optimal operating location, NASA may be able to increase the speed at which they hit minimum success and allow them to run experimental missions that generate more data.

10.4 Marketing Strategies

The most important aspect of our system is its mobility and that will be the prominent design feature seen in our marketing. Our marketing must be done fairly personally by contacting space research and nanosatellite companies. One of the most stressful times of the mission is early orbit where the nanosatellite's location is uncertain. In some cases biology is aboard the spacecraft making early contact very desirable. By increasing the amount of contact time with the satellite the mobile ground station can help with satellite deconfliction and potentially make contact earlier than a static station may be able to.

10.5 Production Cost and Price

The cost of the entire system is approximately $25,000 including the cost of the trailer. This is a one-time investment and the only additional costs required is maintenance of the system. Since the system will be rented along with personal to run the contacts, maintenance can be factored in to the contracting costs and reduced the business' overall investment.

Table 10.1: Price Points For Contracting Mobile Communication Station

Price Point	Beacon	S-Band	Early Operations Only	Full Mission Duration
1	x		x	
2		x	x	
3	x	x	x	
4	x			x
5		x		x
6	x	x		x

The cost of the renting the system will vary on the amount of time the system will be used as well as the number of systems they wish to leverage. Table 10.1 displays the different price points for various combinations of options. The lowest price point utilizes the only the beacon capabilites during early missions ops while the highest price point exercises all the systems on the mobile ground station throughout the entire mission duration.

10.6 Service

The service and maintenance necessary for the mobile ground station will be handled by the operators of the system. If any piece of hardware or software fails the documentation provided for the system will help them replace or fix the broken component. Our system was built robust enough to withstand the elements and the software has been tested for bugs to ensure that failure is at a minimum. Since this is a rented system it is important that regular maintenance is done so that it performs as the customer expects.

Chapter 11

Engineering Standards and Realistic Constraints

11.1 Ethical Analysis

Constructing a mobile satellite communication station to operate nanosatellites poses many ethical questions. Our design team must ensure that we are working ethically with respect to Santa Clara University, the Robotic Systems Laboratory, and NASA. The two primary ethical concerns of our project is the safety of assembly, stowage and delivering a product that is capable of all the functionality specified in our delivery document.

With respect to our customer and users of our system, we are obligated to deliver a safe system that does not violate security rules. Furthermore, the system must operate and contain the same functionality that will be specified in our final design document. Any performance research that is provided with our design should also be clear and accurate. If performance data cannot be generated in an accurate and reliable fashion, then it should not be provided.

Creating a safe system to assemble is also a primary ethical concern for our design team. The project potentially involves moving large components around and assembling them. The requirements of the project specify that two people should be able to accomplish this task. Designing a system with weight specifications for two people will help meet the design requirement and ethical standard.

There will be legal restrictions of our design, the main source of these restrictions

being a product of operating NASA satellites. Data security is the biggest concern. The data paths must be secure to NASA standards and must be verified with the engineering teams that plan to leverage our system. Also, foreign nationals are not allowed to operate our consoles. Since everyone on our design team is a U.S. citizen, this legal concern is outside the scope of our design project. However, we must make the entrances to our system secure so that only authorized personnel are able to have access to the equipment.

There are always social issues when transmitting and receiving radio frequencies. For example being able to transmit at television frequencies would violate the privacy of many households. The Federal Communications Commission (FCC) has specific rules and regulations concerning what frequencies we are able to transmit and what locations we are able to transmit. However this will not be our team's responsibility to ensure our system follows these regulations. We will provide the capabilities to do so, but the customer will need to make sure they are transmitting legally.

The documentation of how we built the system and its operation should be extensive. Since the system may want to be reconstructed and operators other than the design team will be using it, it is important that thorough documentation is kept. All accurate information that we collect during the design process should be shared with the customer. Keeping this documentation will fulfill our ethical responsibilities for subsequent teams that will use or modify our system.

Developing a safe way to assemble our system was a large part of our functional design as well as our ethical obligation. A few of the components will be over 50 lbs and will need to be lifted to a height over 5 feet. We created mechanisms in order to reduce the amount of lifting of each component will help ensure the safety of our system users.

If our project lacks the proper security measures, there is potential for destructive consequences. If an unauthorized user of our system somehow gains access to our command terminals, they may be able to transmit commands to the nanosatellites and compromise their health. Although our technology could not be used to harm humans, it is possible that millions of dollars could be lost to improper use of our system. Several levels

of security will be implemented in our design to ensure that this never occurs.

11.2 Economics

Generally speaking our project, the mobile nanosatellite communication station, has a very large budget when compared to most other projects. Despite this, the capabilities our mobile communication station can offer relative to the cost is very good. Other mobile stations cost thousands of dollars more that our system, and provide the same or, on some occasions, a decrease in performance. The largest cost for our system is the mobile home in of itself. By properly weighting the components of each RV for the cost we are able to maximize the space for the communication workstation as well as the living quarters. In addition, the electrical engineers were able to creatively design a new radio solution that increases overall performance and was able to save a couple thousand dollars when compared to competitor radio solutions.

With this, our project was actually driven from the notion of creating a cheap way for NASA AMES Research Center to track and communicate with nanosatellites in orbit. So therefore, each design decision between performance and cost is critical in the conclusive completion of the mobile nanosatellite communication station.

11.3 Manufacturability

Manufacturability is important to our team's design of the stowing and deployment mechanisms of our 3-meter parabolic satellite dish antenna. Government issued restrictions constrains the design of our vehicle and uphold engineering standards that we must follow. However, most restrictions affect the performance of the vehicle. Since we won't be modifying those components of our RV we can set these restrictions aside. We must follow the height and weight requirements of RV's both for the safety of our operators and in order to get our vehicle approved. According to government-regulated sanctions, our RV cannot exceed the height of 13 ft. Despite the fact that these regulations vary from state to state, we've constrained our design to the minimum requirement (due to the fact that we do

not know where our missions will take us). As for the weight, depending on our expected total weight of the vehicle, some states require particular brakes installed. To avoid this issue we plan to cap our weight depending on the type of trailer we end up purchasing.

11.4 Legal

The designed system as a whole presents two major potential issues related to compliance with government regulations in the matter of public safety: the external modifications to a vehicle (RV) that will be circulating, and the RF subsystems (in special the SDR). Assessing possible infringements in any of these areas is a key aspect because violation of federal or state regulations could jeopardize the imminent missions/operations for which the use of the mobile ground station is already scheduled, and prevent from it being deployed in the near future as planned.

For the integration in the vehicle, we seek to do the least amount of physical alterations to the original vehicle and performed simulations in order to verify its operating safety. The alterations to the vehicle comply with the regulations of the Department of Motor Vehicles (DMV) of the State of California. The system is not designed for vehicle operation while assembled, only the permanent hooks to the vehicle chassis are supposed to remain attached to the trailer while the mast, rotors, dish, etc are intended to be dismantled while in movement. This is not only in compliance with regulations, but also because vibrations and high speed air currents may damage the equipment and/or shorten its lifespan.

The biggest challenge however is represented by the RF equipment. Some of the devices on board of the vehicle are capable of receiving and transmitting with relatively high power in a wide spectrum of frequency bands. This may lead to purposed or involuntary access to restricted bands (both in reception and transmission), for which a special license is required or are banned for civilian use. All equipment deployed as part of the system has been FCC-certified by qualified independent labs (in particular the transceiving devices, amplifiers and antennas), except for the circuit boards designed at SCU. While

the HAM Radio has an specific purpose and is restricted to certain frequency range, the SDR equipment can be reprogrammed with any purpose (from TV signal broadcasting to RADAR) and operate in a very large spectrum. Special measures have to be taken in order to prevent unqualified operators or external personnel to modify the preprogrammed software that governs the SDR subsystem and has been subject of development within this project.

The frequency used by the S-band system also requires special licensing from the FCC in order to transmit. The license is site specific and requires the operations team to pre-approve a location to transmit at S-band frequencies. In order to ensure that the system is following FCC regulations, operators of our system need to plan in advance the locations they wish to operate the station and request licenses. Failure to do so may result in a fine or further restrictions on RF transmission for the RSL.

11.5 Safety

Aside from DMV regulations for public road operations we also concerned ourselves with the setup/breakdown and operation of all our systems. We do not want future students to be continuing our project development and injure themselves while converting the RV into the mobile ground station. With this in mind, it was imperative that we considered the inclusion of various safety precautions in order to prevent accidents caused even by an untrained user.

On the SDR side regarding safety, we have determined that the risk of electric shock to the user is very minimal since at no point during the setup or operation will the user be exposed to any significant voltages. The only thing that should be addressed is that the system (computer, power sources, and USRP) should be properly stowed before driving the RV to prevent it from shifting or falling during transport.

The S-band subsystem also has been designed for safe assembly and disassembly. Many of the components are very heavy, requiring at least two people to carry them. The system was designed with this in mind, allowing for setup to be done without the use of

ladders and minimal lifting of heavy components such as the rotor and dish antenna. The result of this design not only allows for a quick setup, but ensures the safety of the people operating the system.

11.6 Usability

One of the most important aspects of the mobile ground station is its ease of operation. Since we were required that the station must be capable of setting by as few as two people, we had to keep this in mind while designing the parts. Furthermore, the more straightforward the setup process becomes can vastly improve the ease of use by future students. This was achieved by creating a configuration where the user can almost intuitively know how the parts fit together during the setup process.

Similarly, the SDR subsystem also need to be intuitive and self-explanatory in order for future students to use it without our assistance. This was achieved by incorporating various scripts and block diagrams into the design of our system. With the inclusion of the block diagram, turning on or off different blocks becomes very simple. Users can also add their own blocks in future iterations of the SDR design. We also wrote batch files that automatically run all terminal commands in the correct sequence. The batch file helps the user save time as well as make creating future builds of the system very simple.

Chapter 12

Summary

12.1 Project Summary

The mobile ground communication station was designed to increase the success of nanosatellite operations for the RSL. The station achieves this primarily through its mobility, S-band capabilities, and amateur beacon reception capabilities. The mobility of the S-band system is one of the more attractive features of the station, allowing it to relocate to an optimal location for a satellite orbit. The mobile ground station has been found to increase contact availability by over 70% just by operating as another static facility at the same latitude as the current RSL station. With the mobility advantage, availability has the potential to dramatically increase. Increasing contact availability helps ensure minimum success criteria is met for the nanosatellite missions and allows for more complicated missions to be run in the future. Furthermore it can help other nanosatellite projects by helping the satellite deconfliction process in early orbit operations.

The software-defined radio presents a novel approach to beacon reception. It allows for simultaneous multiple satellite accommodation which will help the RSL cope with the increasing number of satellite missions it is being contracted. This system can be automated to track satellites. In addition, it offers a more compact, power-saving and upgradeable solution than a traditional dedicated system. Its functionality can be increased in the future to replace other subsystems like the S-band MicroHard or add new features with a marginal increase in costs.

The auto-calibration system as well as the stowing mechanisms of the dish antenna

enable maximum mobility for the ground station. The setup and breakdown times are under seven hours making operations as convenient and quick as possible. The auto-calibration system ensures that the rotors are tracking with maximum accuracy by compensating for the orientation and location of the vehicle. This system was designed so that the operators did not have to spend extra hours ensuring proper calibration of the dish.

12.2 Future Development

There are several areas for future development of the mobile ground station, the first being improved tracking control for the rotors. Currently the rotors take an azimuth and elevation as input and point in that direction, with a new angle set point occurring every 5 or ten seconds. However this causes considerable vibration throughout the attached dish antenna which increases the amount of maintenance needed in the long run. To fix this problem, the dish can be given a path or a continuous stream electrical input to track the satellite. The rotor then is not moving a degree at a time but rather at an incremental rate that increases and decreases at different times of the pass.

Another potential area of development is using the software-defined radio for S-band communication. The flexibility of the radio solution opens up new S-Band functionality such as recording incoming radio signals and decoding post-pass. Using the SDR may also help determine in early orbit whether the dish antenna is tracking ahead the satellite or behind it as well as helping determine the satellites' identities. Other features like communications in other bands or real time multimedia or IP transmissions may be added without a major economical impact by repurposing the current SDR hardware subsystem.

The SDR subsystem itself will require further fine-tuning as real-life conditions under a mission deployment starts. Current USRP hardware is capable of one simultaneous frequency usage for communications in transceiver mode. It may be upgraded to higher-end models which could enable up to 4 simultaneous channels to be used, using a single computer.

The last area of development is dynamic auto-calibration. With dynamic auto-

calibration the station would theoretically be able to operate while on the move. This would require a different sensor package with more accurate measurements and quicker settling times. Although this may not be useful for S-band communications due to the weight of the satellite dish, the 7-element Yagi antenna can be easily run while on the move.

References

1. C. Kitts, M. Rasay, I. Mas, P. Mahacek, G. Minelli, J. Shepard, and J. Acain, "Responsive Small Satellite Mission Operations Using An Enterprise-Class Internet-Based Command and Control Network." in Proceedings AIAA Space Conference & Exposition, pp. 1-10, San Diego, CA, 2008.
2. C. Kitts, K. Ronzano, R. Rasay, I. Mas, P. Williams, P. Mahacek, G. Minelli, J. Hines, E. Agasid, C. Friedericks, M. Piccini, M. Parra, L. Timucin, C. Beasley, M. Henschke, E. Luzzi, N. Mai, M. McIntyre, R. Ricks, D. Squires, C. Storment, J. Tucker, B. Yost, G. Defouw, A. Ricco, "Flight Results from the GeneSat-1 Biological Microsatellite Mission," in Proceedings AIAA/USU Conf. on Small Satellites, pp. 1-11, Logan, UT, 2007.
3. C. R. Johnson, W. A. Sethares and A. G. Klein, Software Receiver Design: Build Your Own Communication System in Five Easy Steps, 1st ed. Cambridge, U.K.: Cambridge University Press, 2010.
4. D. M. Jansky, M. C. Jeruchim, Communication Satellites in the Geostationary Orbit, 1st ed. Dedham, MA: Artech House, 1983.
5. D. N. Baker, and S. P. Worden, "The Large Benefits of Small-Satellite Missions," Transactions American Geophysical Union, vol. 89, no. 33, pp. 301-302, 2008.
6. E. Grayver, Implementing Software Defined Radio, 1st ed. New York: Springer, 2012.
7. E. Rodrigues, A. Rocha, D. Adrian, "Using GNURadio for a Satellite Beacon Detector" in Proc. 15th Ka and Broadband Communications Navigation and Earth Observation Conf., pp. 152-160, Cagliary, ITA, 2009.
8. GATR Technologies. (2014, May 18). GATR Technologies Manufacturer of Inflatable Portable Satellites in Ku C and X Bands. Available: http://www.gatr.com/products/2-4-antenna-system
9. GlobeComm. (2014, May 18). Explorer Mobile & Transportable Earth Stations. Available: http://www.globecommsystems.com/government/explorer-mobile-earth-stations.shtml
10. J. Breeds, The Satellite Book: A Complete Guide to Satellite TV Theory and Practice, 1st ed. Cricklade, England: Swift Television Publ., 1995.

11. J. Snyder, D. Seeralan, S. Sayed, et al., “Open source software-defined radio tools for education, research, and rapid prototyping,” Int. Journal on Software Tools for Technology Transfer, vol. 16, no. 1, pp. 67-80, 2014.

12. L. Erich, D. Cygan, M. Dippold, F. Dolainsky, and W. Papke, “The Land Mobile Satellite Communication Channel - Recording, Statistics, and Channel Model,” IEEE Transactions on Vehicular Technology, vol. 40, no. 2, pp. 375-86, 1991.

13. L. G. Reis, A. F. Barros, K. G. Lenzi, P. Meloni, S. E. Barbin, “Introduction to the Software-defined Radio Approach,” Latin America Transactions, IEEE (Revista IEEE America Latina), vol. 10, no. 1, pp. 1156-1161, 2012.

14. M. Cheffena, L. E. Braten, “LowCost Digital Beacon Receiver Based on Software-Defined Radio,” Antennas and Propagation Magazine, IEEE, vol. 53, no. 1, pp. 50-55, 2011.

15. M. De Milliano, C. Verhoeven, “Towards the next Generation of Nano Satellite Communication Systems,” Acta Astronautica, vol. 66, no. 9, pp. 1425-433, 2010.

16. R. Sousa, J. Pires, A. Rocha, “DSP based software/digital PLL detector for satellite beacon receivers,” Signal Processing and Its Applications, 2007. ISSPA 2007. 9th International Symposium, pp. 1,4, 12-15, Sharjah, United Arab Emirates, 2007.

17. U. Kuhar, G. Kandus, A. Vilhar, “Lowcost frequency stable beacon receiver based on software defined radio,” Software, Telecommunications and Computer Networks (SoftCOM), 2012 20th International Conf., pp.1,5, Split, Croatia, 2012.

Appendix A

Product Design Specification

Table A.1: Customer Needs Matrix for the Mobile Ground Communication Station

Requirement	Units	Design Target	Additional Notes
System Relocation	-	Drive-able Distances	
Minimum Personnel Required	people	2	Safe setup by two
Internet Access	-	Yes	Method for accessing network
Living Quarters	-	Yes	Large enough to house two
Power Source	-	Yes	Off-grid
Rugged	-	Yes	Must withstand elements
Operating Frequency (RX)	GHz	2.4-2.483	-
Operating Frequency (TX)	GHz	2.4-2.483	-
Elevation Adjustment Range	Degrees	0-	-
Azimuth Adjustment Range	Degrees	0-90	-
Self-Contained	-	0-360	-
Contact Availability Increase	percent	60	-
Pointing Error	Degrees	<1	-
Collapsed Volume	m^3	1.6	-
Setup Time	hours	24	-
Breakdown Time	hours	24	-
Beacon Decode	percent	80	-
SDR Price Reduction	U.S. dollars	50	-
Total System Cost	U.S. dollars	25,000	-
Sensor Information Distribution	-	Every subsystem	All subsystems have access
Auto-calibration Senstivity	degrees	<1	-
Max Wind Speed Operation	m/s	14	-

A.1 Questionnaires

Questionnaire 1
1. What are critical(primary) requirements?
2. What are recommended(secondary) requirements?
3. What are suggested(ternary) requirements?
4. What hardware MUST be used?
5. What systems can be modified?

6. What software can be used?
7. How do devices need to be powered?
8. Does it need to have internet access at all times? If so, how do we handle security?
9. What would be convenient features for the vehicle?
10. How long would you be willing to set up the system?
11. Would it be okay if there was a user manual? Or should the system be intuitive enough to not need a manual?
12. Can the stowing process take longer than the setup? If so how long can the stowing process take?
13. How many people must it seat?
14. What materials can we use/are we allowed to use?
15. What frequencies must we support?
16. Delivery date?
17. What missions will it be used for?
18. How much space are ops equipment allotted?
19. Will other systems use the same workstations?
20. What radio frequencies should we expect to be able trancieve and recieve?
21. Size limitations?
22. How heavy can the system be?
23. Duration of missions?
24. Possible mission locations? Any severe weather to be encountered?
25. Do you have any experience using any other satellite mobile communication stations?

Questionnaire 2

1. If you had to imagine nanosat operations being done differently than they currently are, for example different systems or capabilities, what would you think of?
2. So we are currently in the process of constructing a mobile operations station to support the nanosat missions. Do you have any experience using any other satellite mobile communication stations?
3. Can you think of any potential issues when creating a mobile com station, from the perspective of NASA engineers as well as the perspective of SCU?
4. Security is always a concerning issue when it comes to operating these satellites. What security measures would we need to take to satisfy the NASA security standards when operating a mobile communication station?
5. When you contract teams to operate your nanosatellites, what are the top three features are you looking for?
6. How valuable would a mobile communication station be when choosing a nanosatellite operations team?
7. Is it possible that with a mobile station SCU would be chosen over another operations team for a mission?
8. From the previous interaction you have had with the SCU operations team, is there an area of how we run operations that we can improve on?

A.2 Budget

Table A.2: Complete Cost Report for Mobile Communication Station

	Budget Update				
TEAM	Mobile Satellite Communication Station				
Date	5/8/2014				
INCOME					
Category	Source	Sought	Committed	Pending	
Grant	Undergraduate Programs Senior Design Project Fund	$1500	$1500	$0	
	SCU Technology Steering Committee	$10000	$10000	$0	
	The Robotic Systems Laboratory	10000	$10000	$0	
	TOTAL	$21500	$21500	$0	$21500
EXPENSES					
Category	Description	Estimated	Spent	Pending	
Operations Vehicle	Rec. Vehicle	$10000	10000	$0	
Independent Workstations	Computers	$900	$900	$0	
	Cabling	$200	$200	$0	
S-Band System	Tracking Computer	$600	$600	$0	
	Data Computer	$600	$600	$0	
	2.4 Meter Dish Antenna	$1396	1396	$0	
	Antenna Feed	$165.71	$165.71	$0	
	MHX radio	$795	$795	$0	
	SPID RAS Rotor and Controller	$2129.57	2129.57	$0	
	Support Structure	$257	$257	$0	
	Misc Hardware	$325.6	$325.6	$0	
Software-Defined Radio	USRP B200	$675	$675	$0	
	7-element Yagi Antenna	$60	$60	$0	
	LNA Preamp (4420B)	$127	$127	$0	
	Cables and Connectors	$40	$40	$0	
	Data Computer	$600	$600	$0	
	Tracking Computer	$600	$600	$0	
	G-5500 Rotor and controller	$749.95	$749.95	$0	
	Arduino Uno	$24.71	$24.71	$0	
Auto-Calibration System	APM 2.6	$129.88	$129.88	$0	
	TOTAL	$20375.42	$20375.42		0
	Net Reserve (Deficit)		$1124.58	$0	$0

Appendix B

Block Diagrams

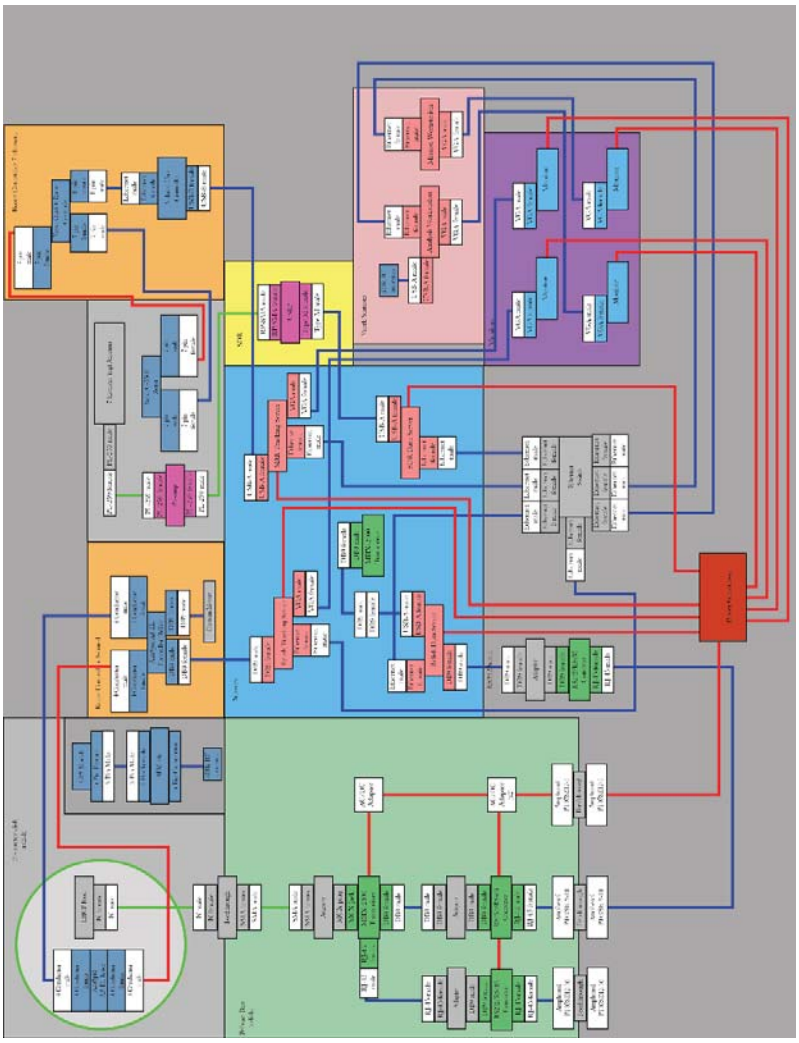


Figure B.1: Mobile Communication Station Block Diagram

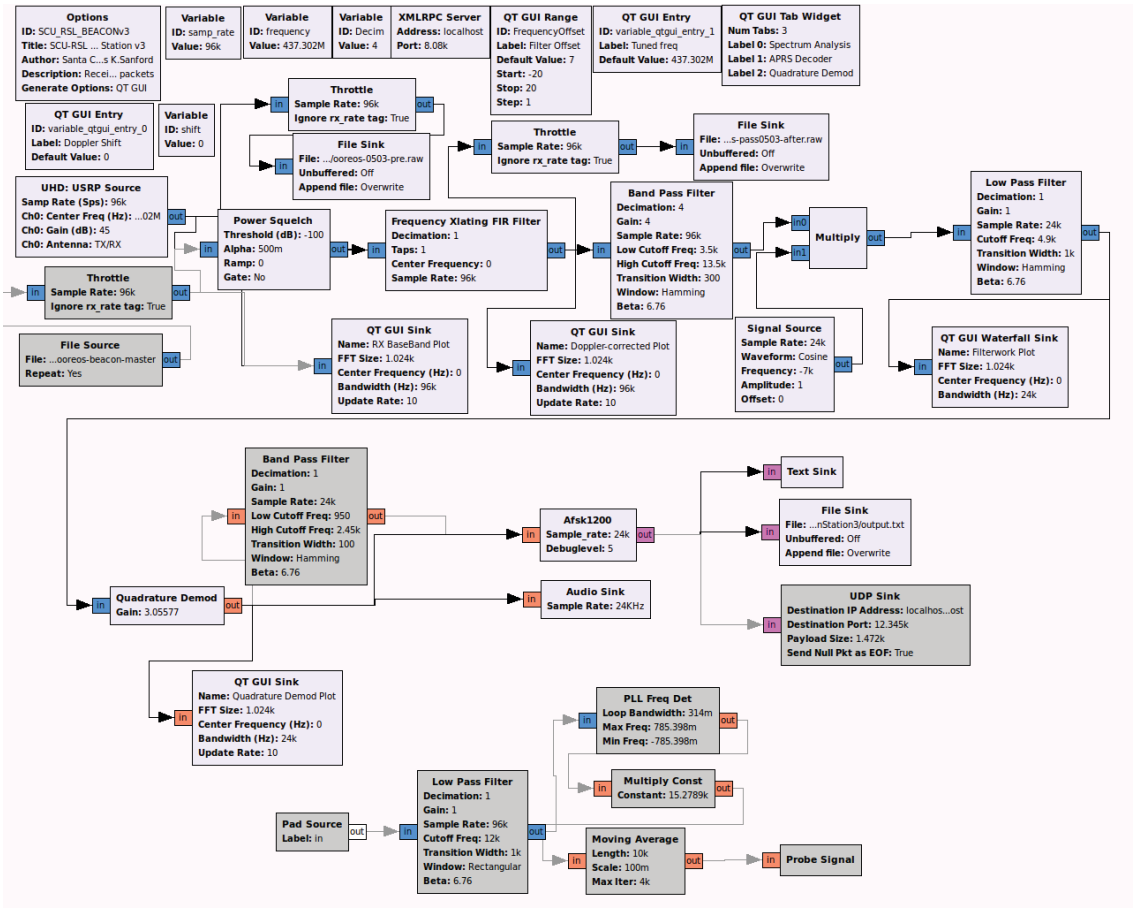


Figure B.2: SDR Block Diagram

Appendix C

Parts List and Design Drawings

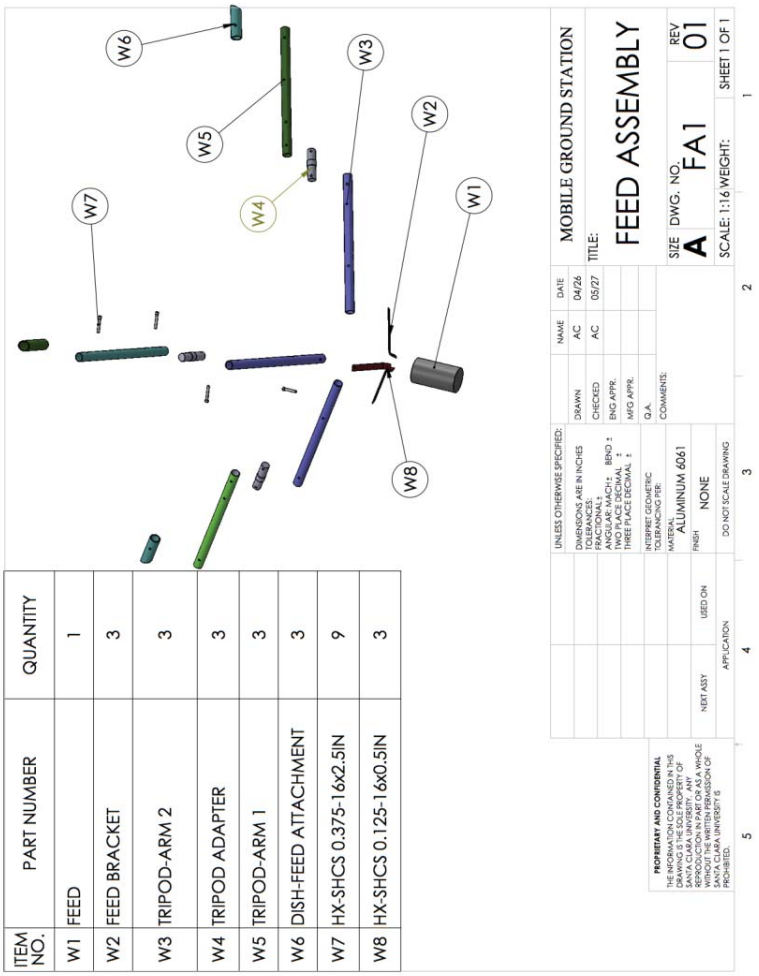


Figure C.1: Detailed Drawings

Figure C.2: Detailed Drawings

Figure C.3: Detailed Drawings

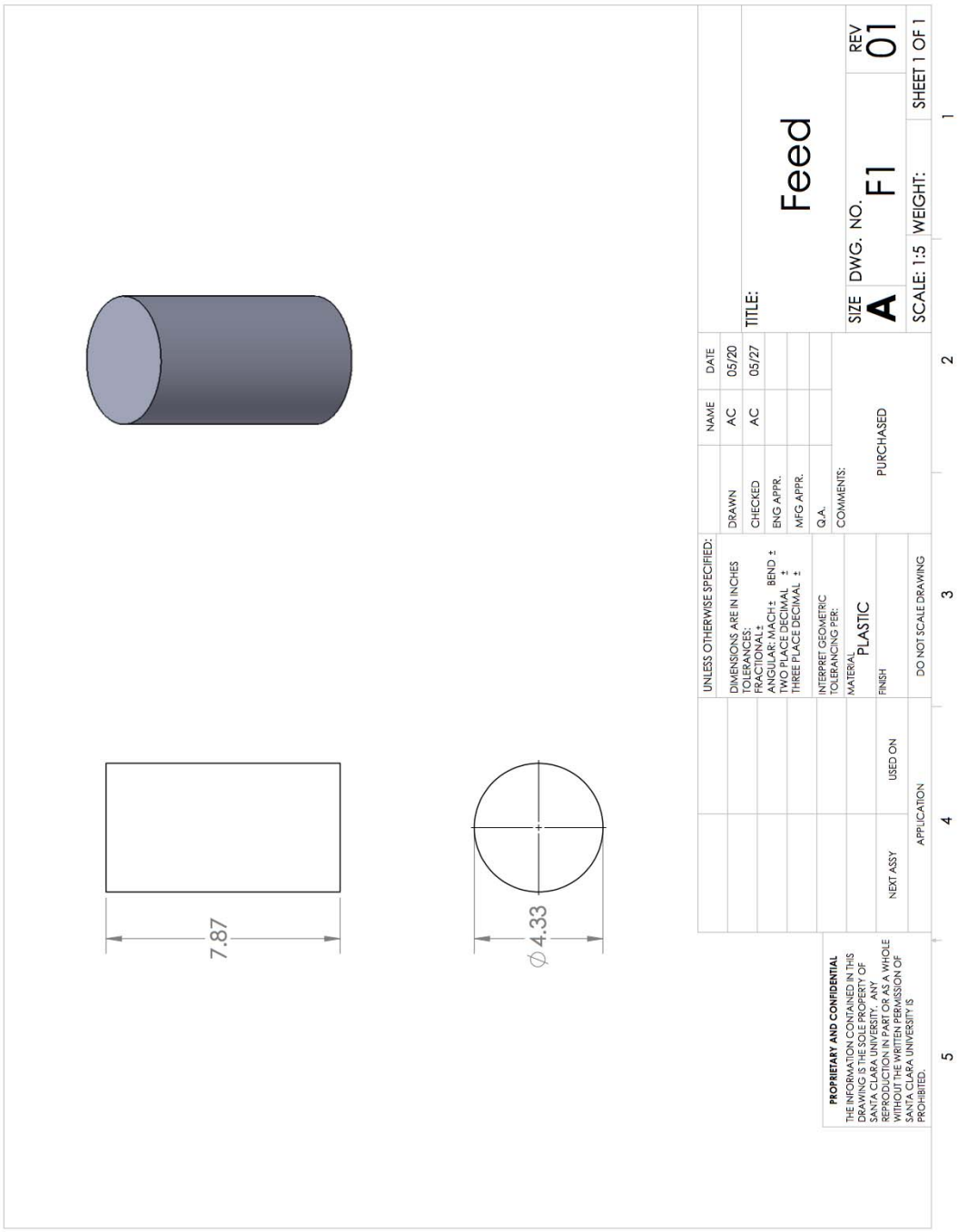


Figure C.4: Detailed Drawings

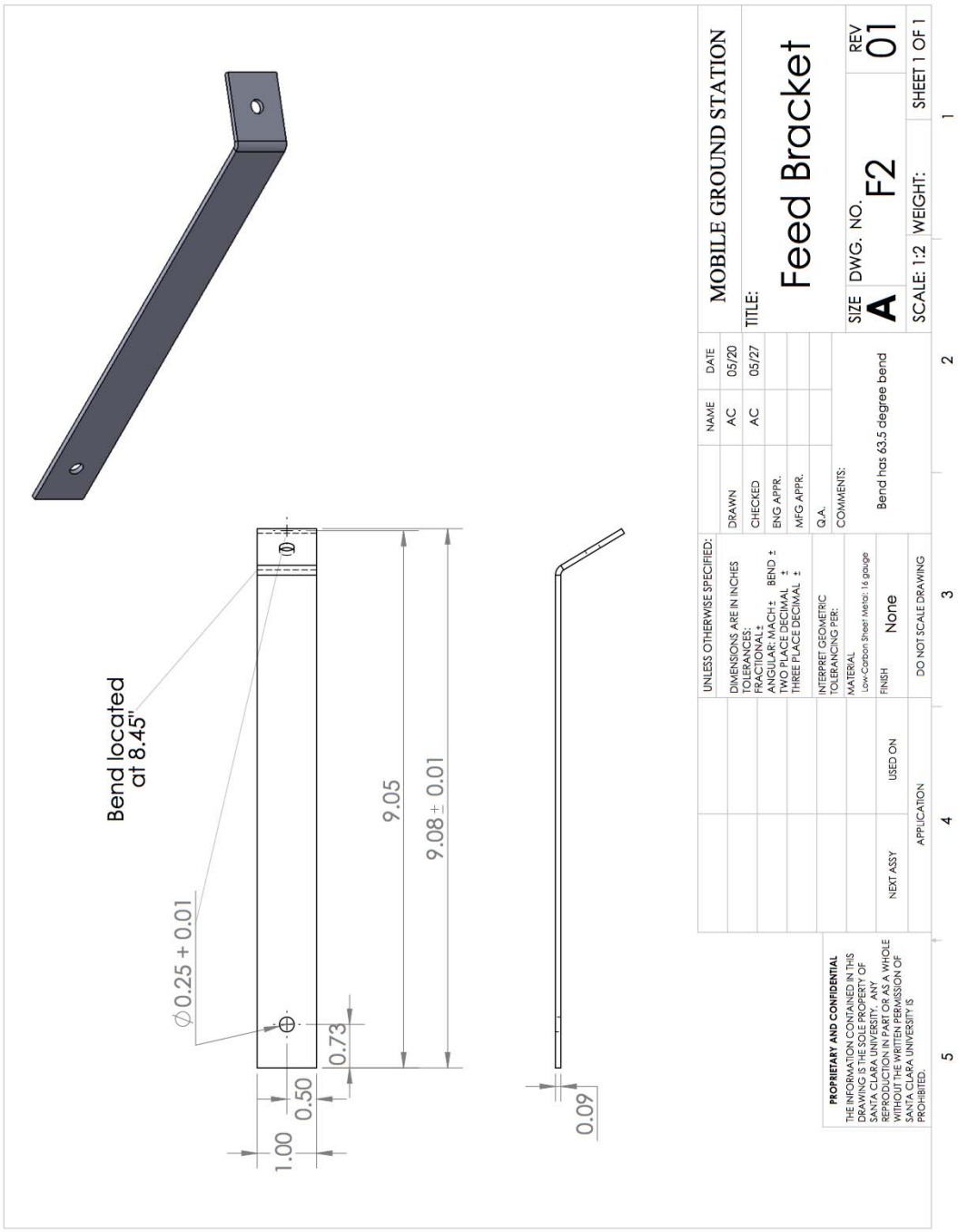


Figure C.5: Detailed Drawings

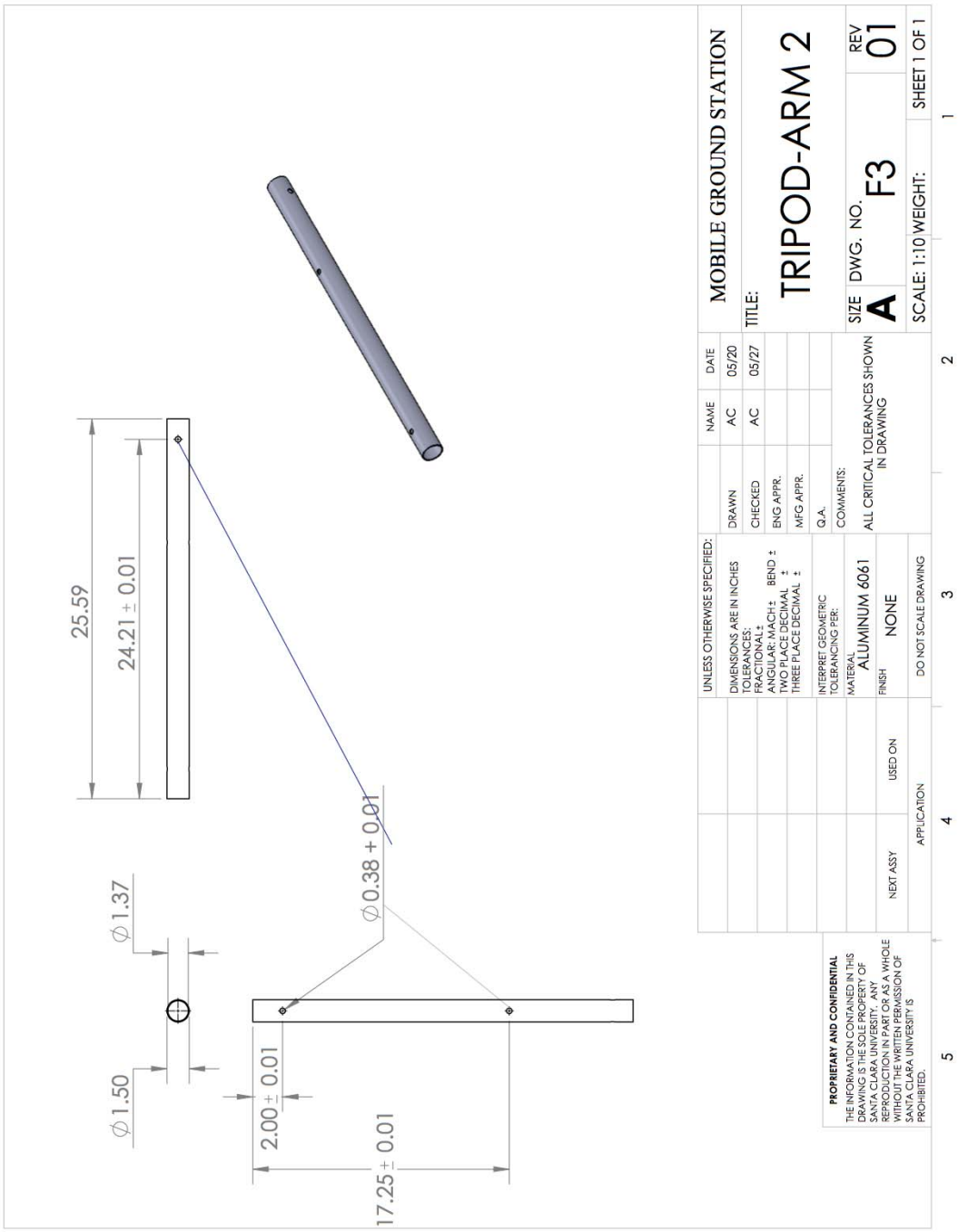


Figure C.6: Detailed Drawings

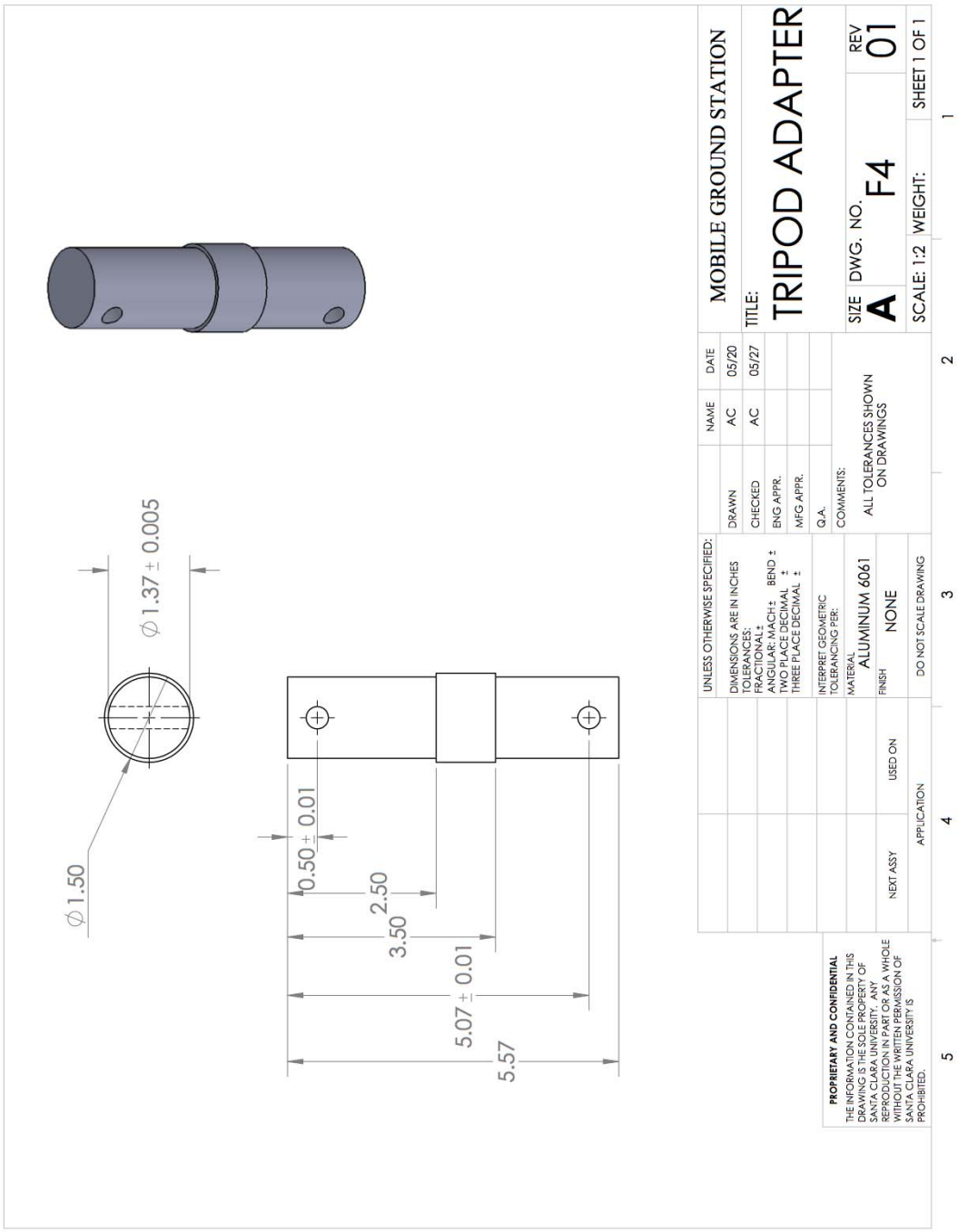


Figure C.7: Detailed Drawings

Figure C.8: Detailed Drawings

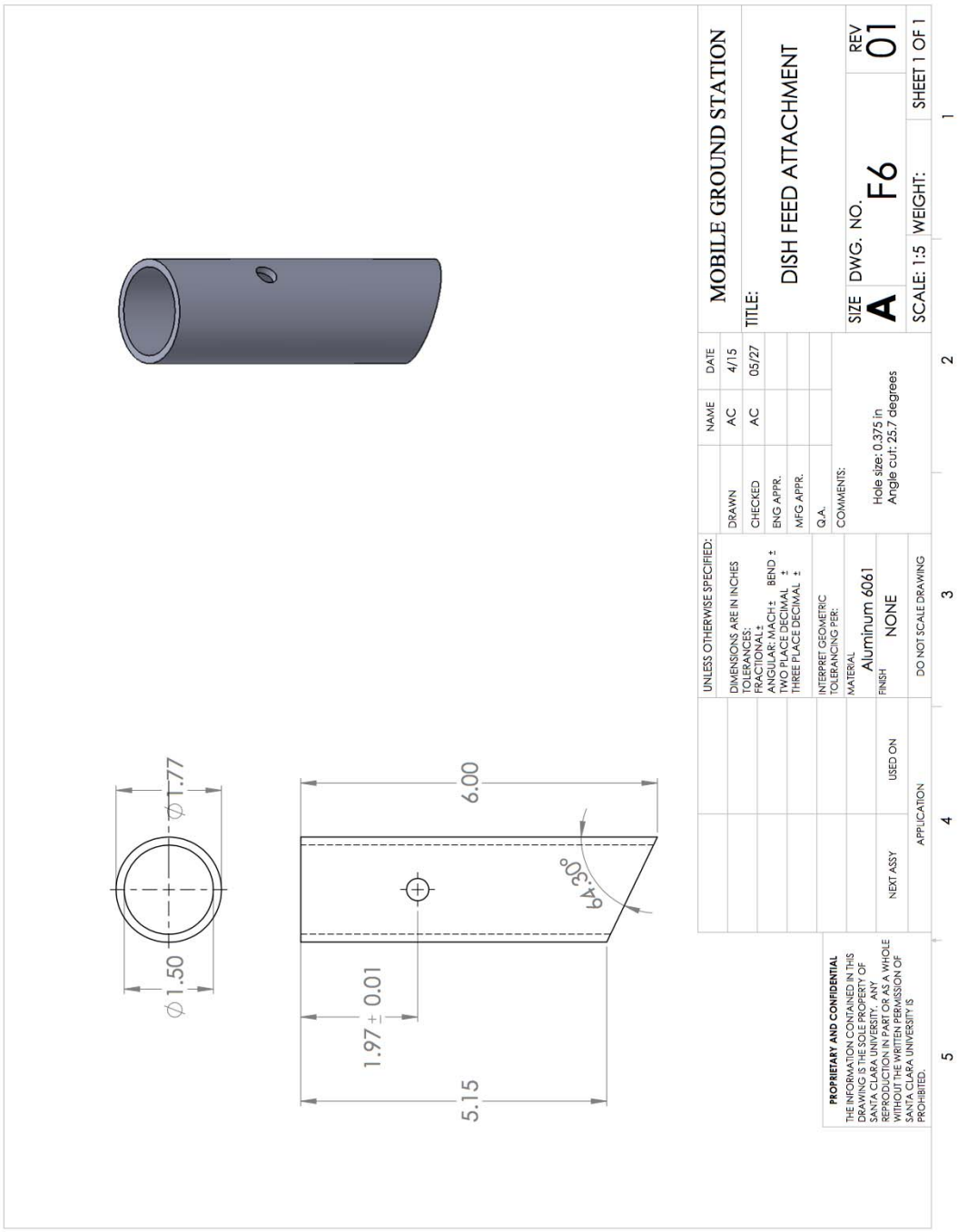


Figure C.9: Detailed Drawings

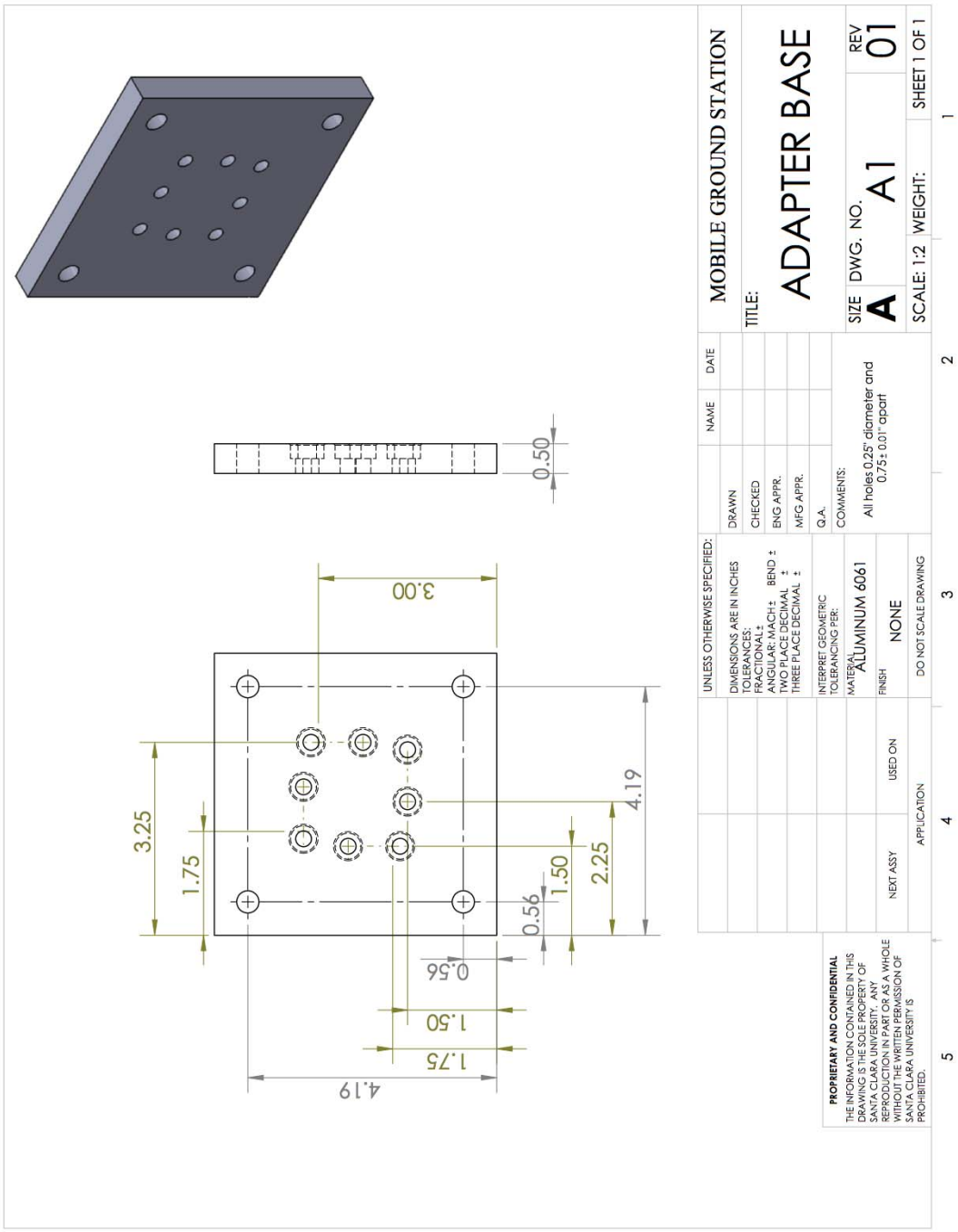


Figure C.10: Detailed Drawings

Figure C.11: Detailed Drawings

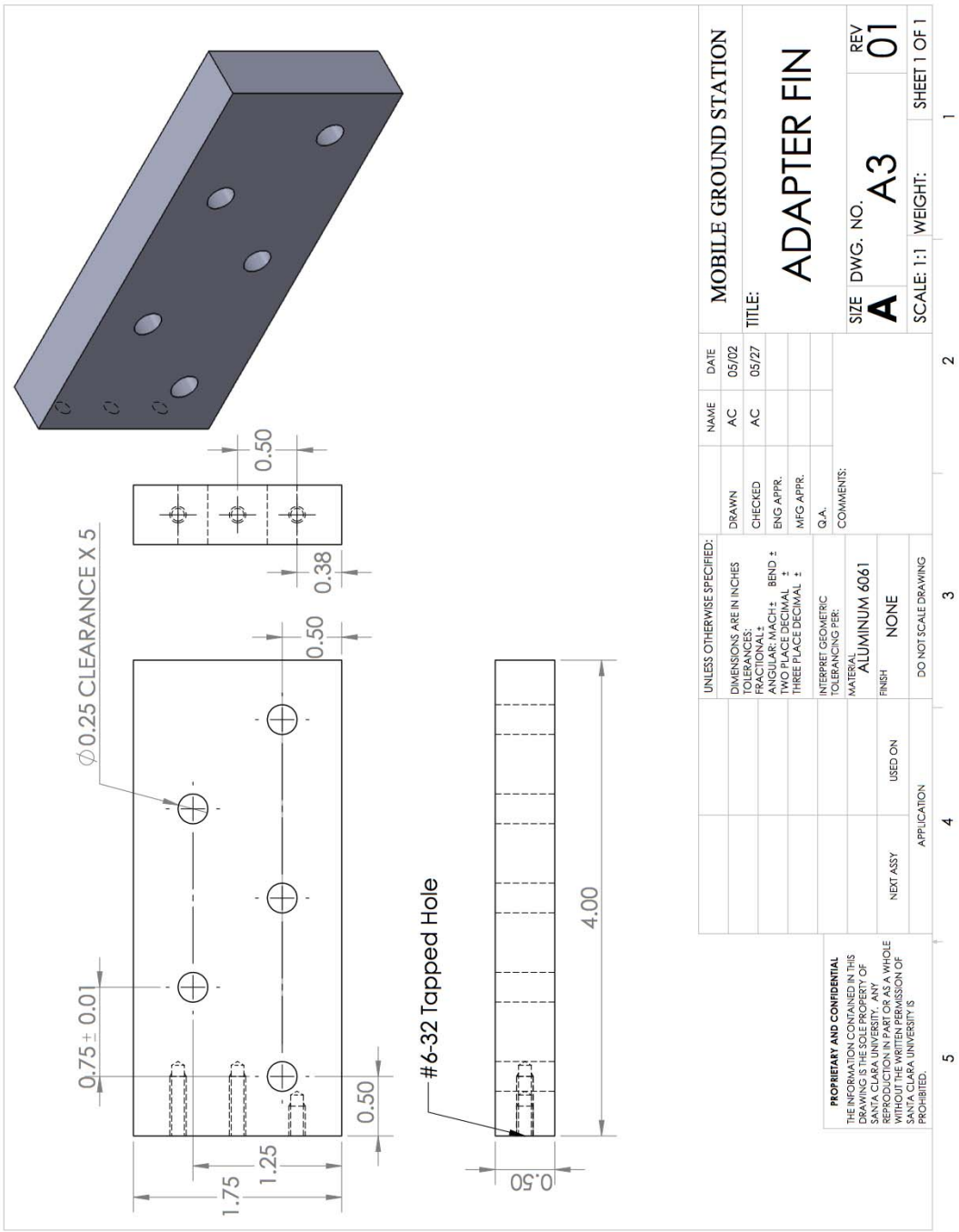


Figure C.12: Detailed Drawings

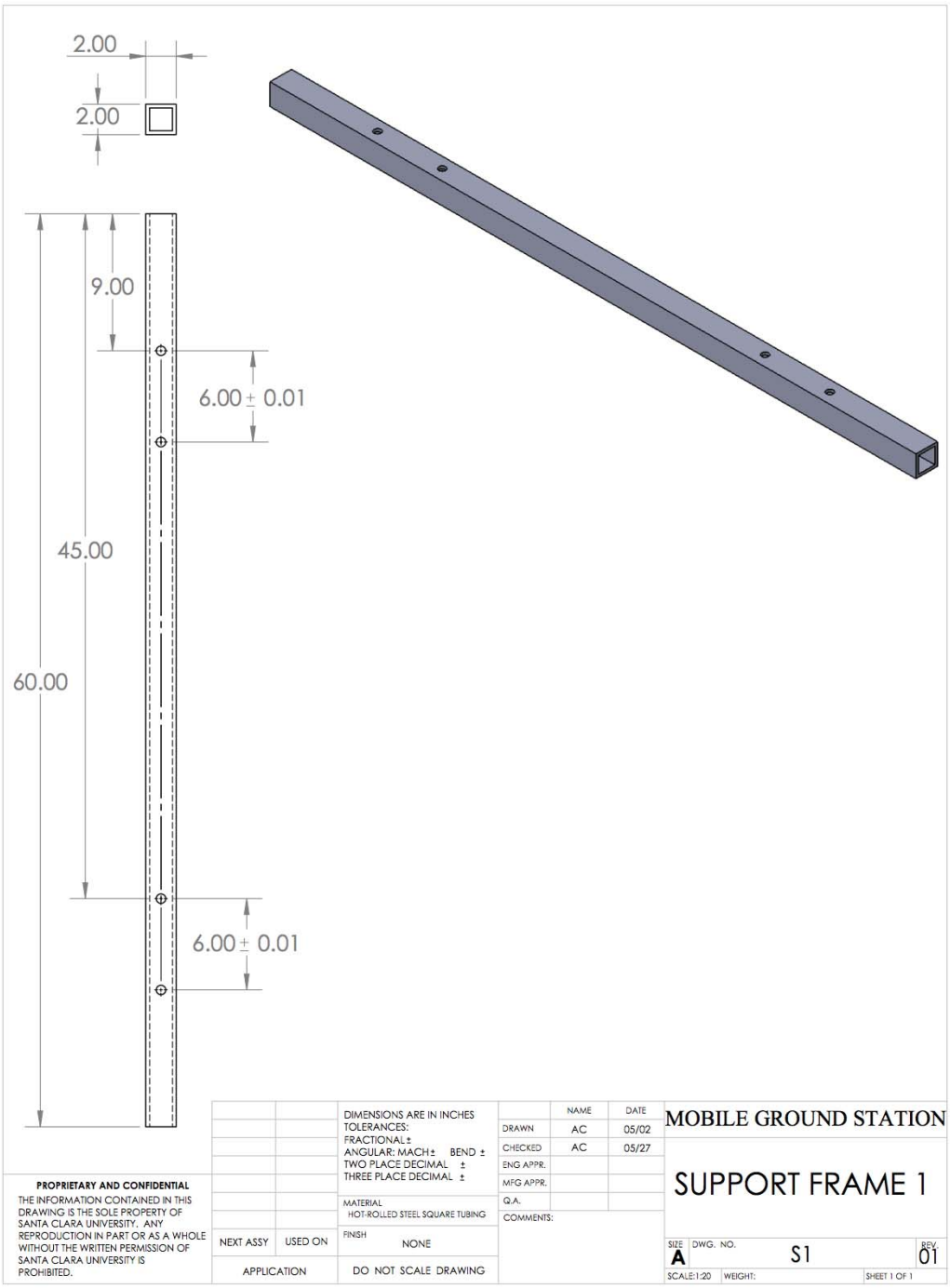


Figure C.13: Detailed Drawings

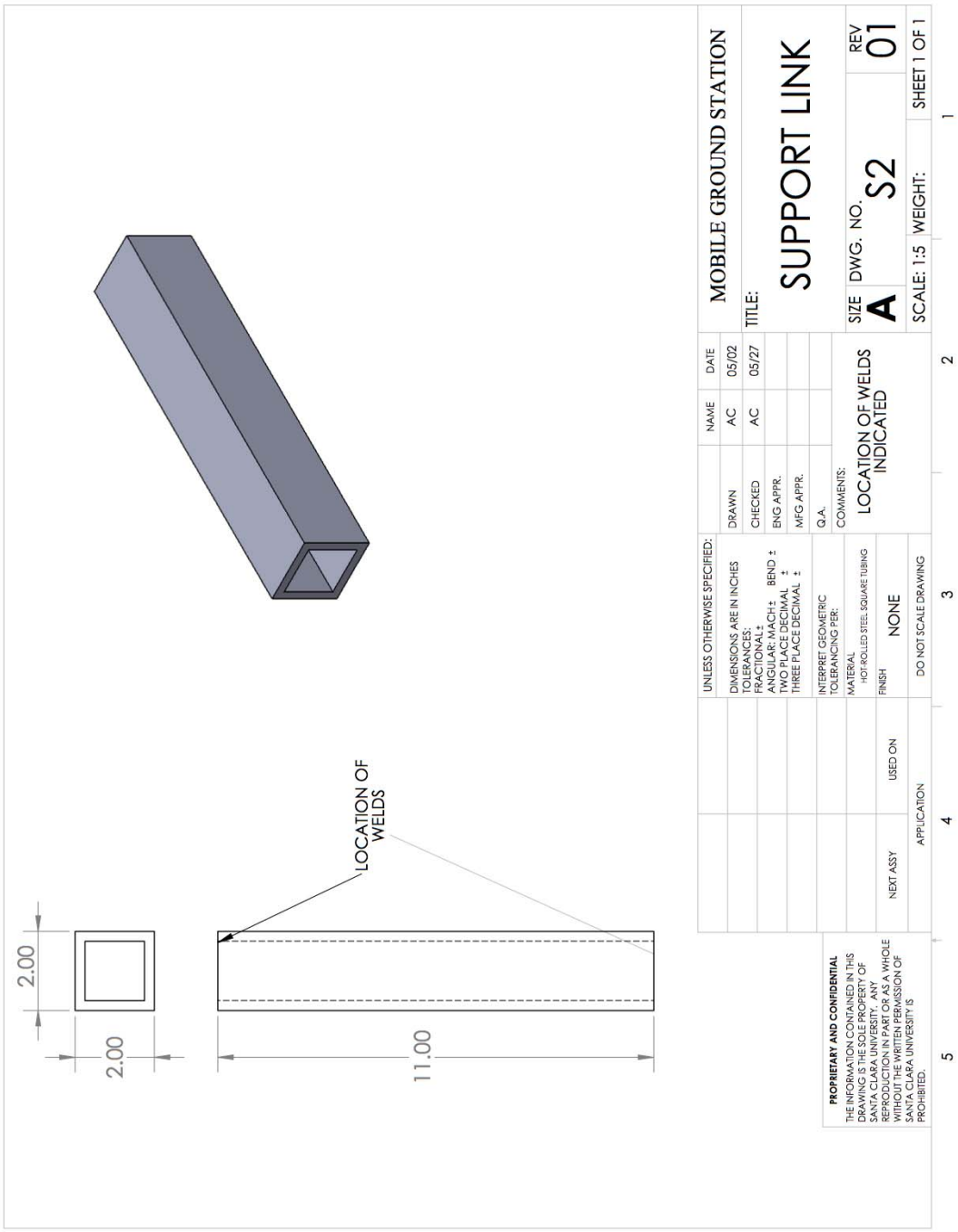


Figure C.14: Detailed Drawings

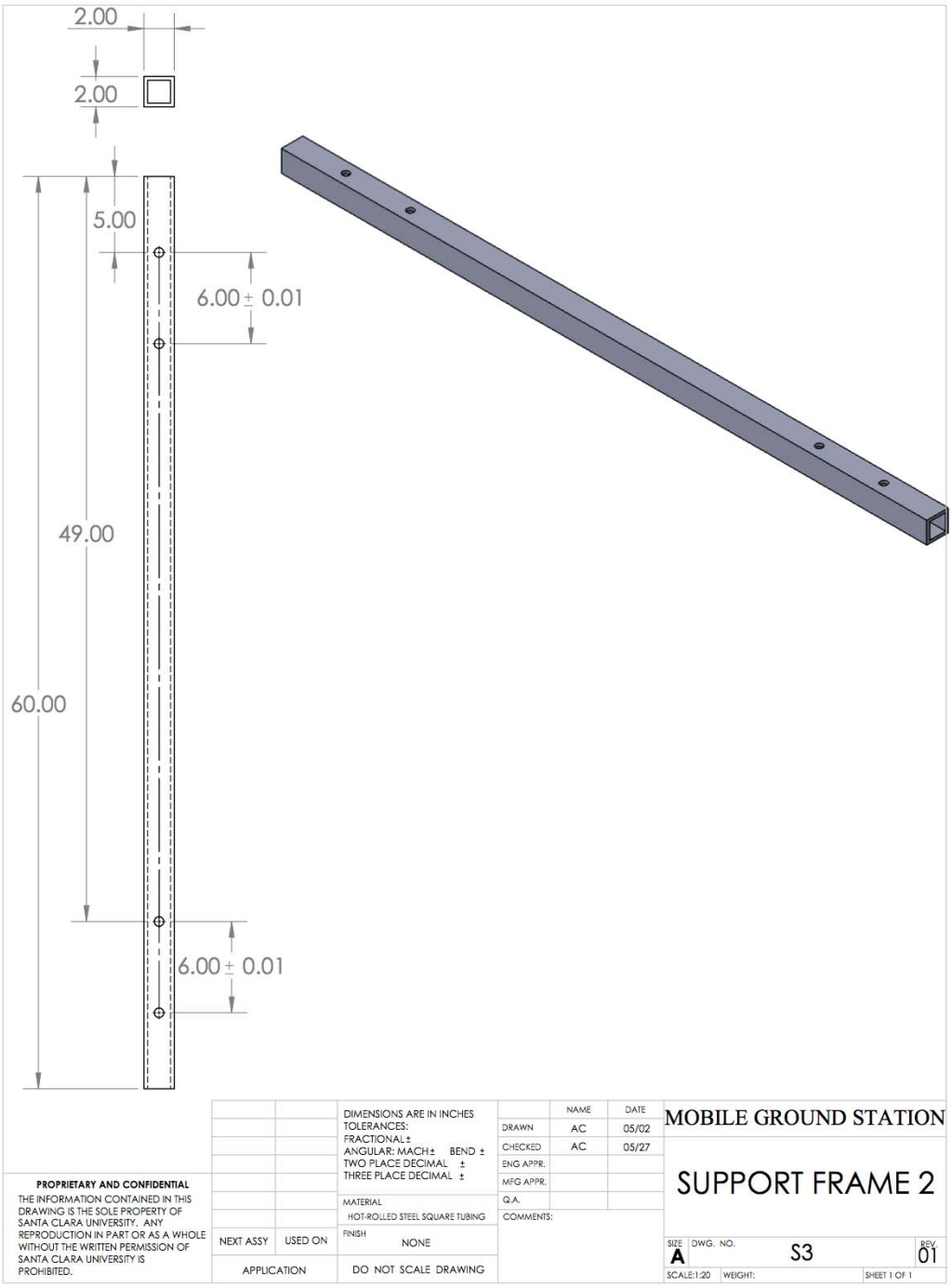


Figure C.15: Detailed Drawings

Appendix D

Specification Sheets

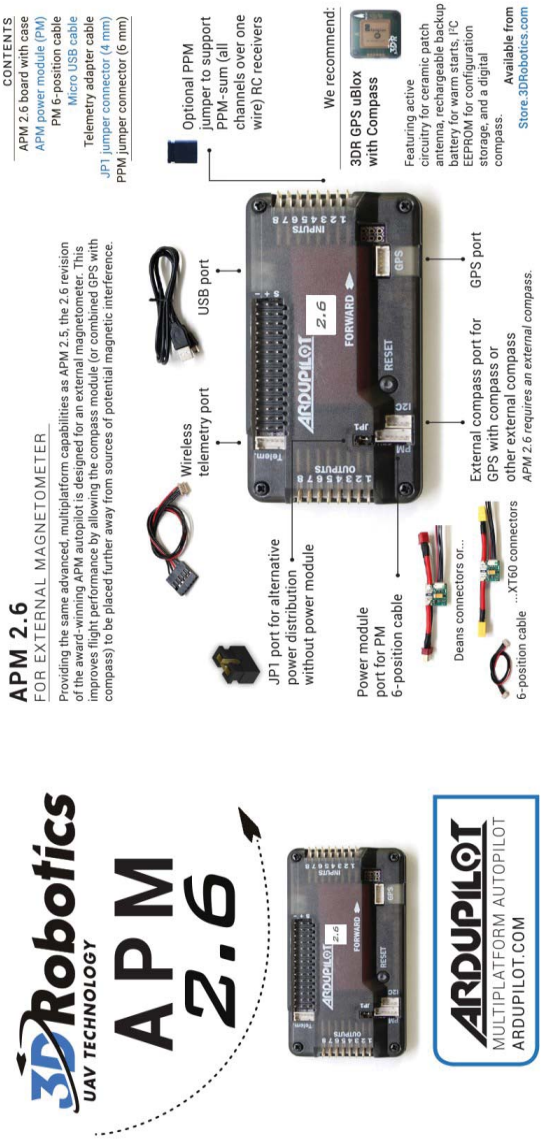

CONTENTS
APM 2.6 board with case
APM power module (PM)
PM 6-position cable
Micro USB cable
Telemetry adapter cable
JP1 jumper connector (4 mm)
PPM jumper connector (6 mm)
APM 2.6
FOR EXTERNAL MAGNETOMETER
Providing the same advanced, multiplatform capabilities as APM 2.5, the 2.6 revision of the award-winning APM autopilot is designed for an external magnetometer. This improves flight performance by allowing the compass module (or combined GPS with compass) to be placed further away from sources of potential magnetic interference.
Optional PPM jumper to support PPM-sum (all channels over one wire) RC receivers
We recommend:
3DR GPS uBlox with Compass
Featuring active circuitry for ceramic patch antenna, rechargeable backup battery for warm starts, I²C EEPROM for configuration storage, and a digital compass.
Available from Store.3DRobotics.com
USB port
Wireless telemetry port
JP1 port for alternative power distribution without power module
Power module port for PM 6-position cable
Deans connectors or...
6-position cable
...XT60 connectors
GPS port
External compass port for GPS with compass or other external compass
APM 2.6 requires an external compass.
3DRobotics
UAV TECHNOLOGY
APM 2.6
ARDUPILOT
MULTIPLATFORM AUTOPILOT
ARDUPILOT.COM

Figure D.1: APM 2.6 Specifications

Ettus Research

USRP™ B200/B210 Bus Series

FEATURES

- Coverage from 70 MHz – 6 GHz RF
- GNU Radio, C, and Python Compatible
- USB 3.0 High speed interface
  - (Compatible with USB 2.0)
- Flexible rate 12 bit ADC/DAC

USRP B200

- 1 TX, 1 RX, Half or Full Duplex
- Xilinx Spartan 6 XC6SLX75 FPGA
- Up to 56 MHz of real-time bandwidth
- USB Bus powered

USRP B210

- 2 TX, 2 RX, Half or Full Duplex
- Fully-coherent 2x2 MIMO capability
- Xilinx Spartan 6 XC6SLX150 FPGA
- Up to 56 MHz of real-time bandwidth 1x1
- Up to 32 MHz of real-time bandwidth 2x2
- Includes DC power supply

B200/B210 Product Overview

The USRP B200 and B210 come straight from the R&D labs of Ettus Research providing early access to cutting edge experimental hardware covering 70MHz – 6 GHz with integrated RFIC technology, a Spartan6 FPGA, and USB 3.0 connectivity. This new platform enables experimentation with wide range of applications including FM and TV broadcast, cellular, WiFi, ISM, and more. The USRP B200 features one receive and one transmit channel in a bus-powered, board-only package. The USRP B210 features two receive and two transmit channels, incorporates a larger FPGA and includes an external power supply. Both use a new Analog Devices RFIC to deliver a cost-effective experimentation platform and a high bandwidth USB 3.0 bus with up to 56 MHz of instantaneous bandwidth on select USB 3.0 chipsets (with backward compatibly to USB 2.0). With this new kit, users can develop their GNU Radio applications and seamlessly port their designs to higher performance USRP systems such as USRP N210 with industry proven 1 Gigabit Ethernet connectivity or the E100 embedded USRP form factor, both including discrete RF boards with higher sensitivity, dynamic range, and IP3 performance using the common USRP Hardware Driver (UHD) framework.

Application Development with the USRP B200/B210

The USRP B200 and B210 is supported by the USRP Hardware Driver™ (UHD) software. UHD is an open-source, cross-platform driver that can run on Windows, Linux, and MacOS. It provides a common API, which is used by several software frameworks, such as GNU Radio. With this software support, users can collaborate with a vibrant community of enthusiasts, students, and professionals that have adopted USRP products for their development. As a member of this community, users can find assistance for application development, share knowledge to further SDR technology and contribute your own innovations. Product support is also available from Ettus Research.

The Analog Devices AD9361 RFIC

USRP B200 and B210 use the AD9361 RFIC from Analog Devices. Ettus Research and Analog Devices have established a partnership to bring this cutting-edge RF technology to the market in the form of an easy-to-use SDR. With the USRP B200 and B210, designers can prototype with the AD9361 quickly and easily.

Figure D.2: B200 Specification Sheet

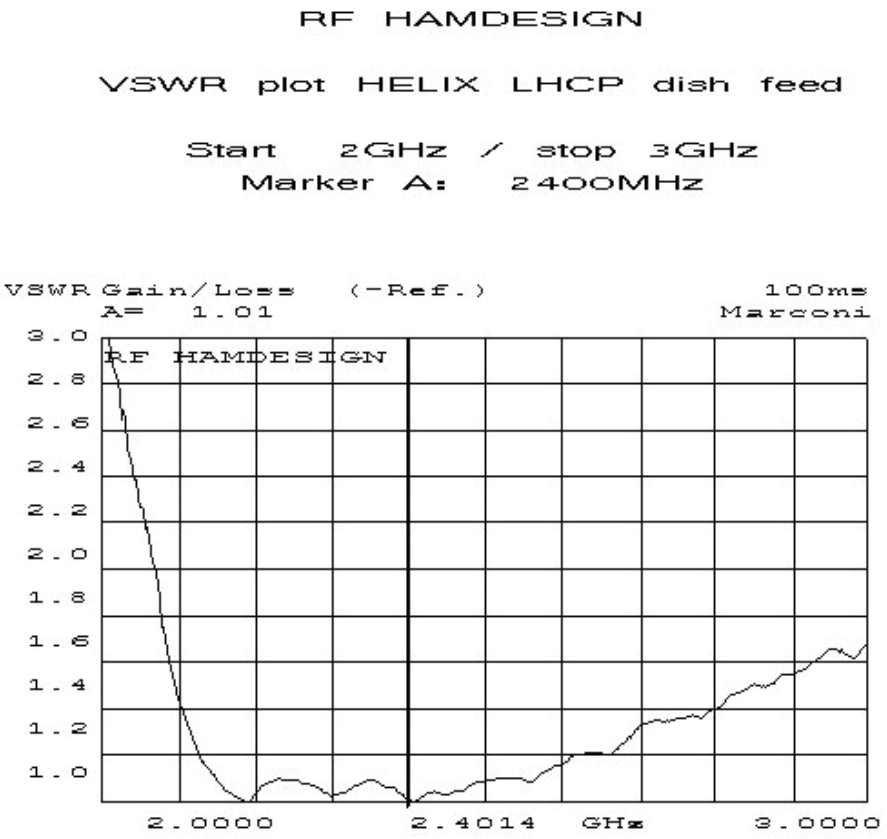


Figure D.3: Dish Feed Frequency Specification

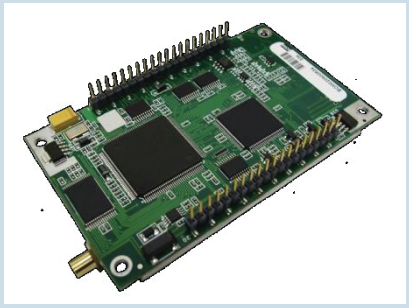
MHX2420

2.4 GHz OEM Industrial Wireless Modem

The MHX2420 is a 2.4 GHz Frequency Hopping Spread Spectrum Modem, which can be optimized for long distance communication over 30 miles (50km). MHX2420 radios offer the fastest communication over the longest distances

Applications

- *SCADA (PLCs, MODBUS), Telemetry*
- *Commercial Communications*
- *GPS Vehicle Data/Tracking, DGPS*
- *Electric, Oil & Gas Sensors/Detection*
- *Display Signs*
- *Transparent Low Latency Communication*



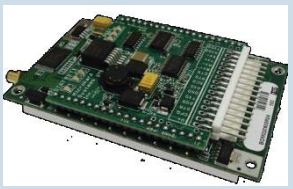
Microhard Systems Inc.'s proprietary radio technology excels in the most demanding RF and physical environments.

The MHX2420 features robust, low latency, secure data communications. It has full serial port and separate diagnostics port for real-time diagnostics without interrupting data communications. Excellent noise figure, superior interference rejection, very agile frequency synthesis, digital modulation, and matched filter detection, are among the many advantages to using the MHX2420.

Features of the MHX2420

- Transparent, low latency link, providing 230 kbps continuous throughput to support protocols such as MODBUS
- Communicates with virtually all PLCs, RTUs, and serial devices
- Industrial Grade - extended temperature specification
- Supports Point-to-Point, Point-to-Multipoint, Peer-to-Peer, Store and Forward Repeater, TDMA, Multimaster
- Maximum allowable transmit power (1W)
- Low power consumption in Sleep Mode and High Voltage Option
- 32 bits of CRC, selectable Forward Error Correction with retransmission
- Separate diagnostics port - transparent remote diagnostics and online network control
- Footprint backwards compatible with MHX2400

MHX2420 HV Option



Rev 3.1

CONFIDENTIAL

Figure D.4: MicroHard Radio Specification Sheet

Alfa Radio Ltd.
11511 - 154 St., Edmonton, Alberta, Canada T5M 1X8
URL www.alfaradio.ca email alfa@alfaradio.ca
Voice 780 466 5779 Facsimile 780 466 4956

Specifications: AlfaSpid RAK Azimuth Rotator	RAK @ 12V	RAK @ 18V	RAK @ 24V
Turning torque in lbs / N m	1,400 in lb / 158 N m	1,800 in lb / 203 N m	3,240 in lb / 366 N m
Brake torque in lbs / N m	14,000 in lb / 1,582 N m	14,000 in lb / 1,582 N m	14,000 in lb / 1,582 N m
Brake Construction	Double Worm Gears	Double Worm Gears	Double Worm Gears
Vertical Load lbs / Kg	550 lbs / 250 Kg	550 lbs / 250 Kg	550 lbs / 250 Kg
Rotation Speed (360 Deg)	120 sec	90 sec	60 sec
Precision	1 deg	1 deg	1 deg
Rotation Range	720 deg	720 deg	720 deg
Weight lbs / Mass Kg	16 lbs / 7 Kg	16 lbs / 7 Kg	16 lbs / 7 Kg
Position Sensor	Reed Switch	Reed switch	Reed Switch
Mast size top / bottom – in / mm	2.0 / 2.6 in, 50 / 66 mm	2.0 / 2.6 in, 50 / 66 mm	2.0 / 2.6 in, 50 / 66 mm

Specifications: AlfaSpid BIG RAK Azimuth Rotator	BIG RAK @ 12V	BIG RAK @ 18V	BIG RAK @ 24V
Turning torque in lbs / N m	>5,000 in lb / 565 N m	>8,000 in lb / 904 N m	Contact AlfaRadio
Brake Torque in lbs / N m	24,000 in lb / 2,712 N m	24,000 in lb / 2,712 N m	24,000 in lb / 2,712 N m
Brake Construction	Double Worm Gears	Double Worm Gears	Double Worm Gears
Vertical Load lbs / Kg	>700 lbs / 318 Kg	>700 lbs / 318 Kg	>700 lbs / 318 Kg
Rotation Speed (360 Deg)	120 sec	90 sec	60 sec
Precision	1 / 0.5 deg	1 / 0.5 deg	1 / 0.5 deg
Rotation Range	720 deg	720 deg	720 deg
Weight lbs / Mass Kg	24 lbs / 11 kg	24 lbs / 11 kg	24 lbs / 11 kg
Position Sensor	Reed Switch	Reed switch	Reed Switch
Mast size top / bottom – in / mm	2.6 / 2.6 in, 66 / 66 mm	2.6 / 2.6 in, 66 / 66 mm	2.6 / 2.6 in, 66 / 66 mm

Alfa Radio Ltd.
11511 - 154 St., Edmonton, Alberta, Canada T5M 1X8
URL www.alfaradio.ca email alfa@alfaradio.ca
Voice 780 466 5779 Facsimile 780 466 4956

Specifications: AlfaSpid RAS Azimuth and Elevation Rotators	RAS @ 12V	RAS @ 18V	RAS @ 24V
Turning torque in lbs / N m	1,400 in lb / 158 N m	1,800 in lb / 203 N m	3,240 in lb / 366 N m
Brake torque in lbs / N m	14,000 in lb / 1582 N m	14,000 in lb / 1582 N m	14,000 in lb / 1582 N m
Brake Construction	Double Worm Gears	Double Worm Gears	Double Worm Gears
Vertical Load lbs / Kg	550 lbs / 250 Kg	550 lbs / 250 Kg	550 lbs / 250 Kg
Rotation Speed (360 Deg)	120 sec	90 sec	60 sec
Precision	1 deg	1 deg	1 deg
Rotation Range Az / El	360 /180 deg	360 / 180 deg	360 /180 deg
Weight lbs / Mass Kg	29 lbs / 13 Kg	29 lbs / 13 Kg	29 lbs / 13 Kg
Mast size inches / mm Bottom	2.6 in / 66 mm	2.6 in / 66 mm	2.6 in / 66 mm
Mast size inches / mm Elevation	2.0 in / 50 mm	2.0 in / 50 mm	2.0 in / 50 mm

Specifications: AlfaSpid BIG RAS Azimuth and Elevation Rotator	BIG RAS @ 12V	BIG RAS @ 18V	BIG RAS @ 24V
Turning torque in-lbs/ N m	>5,000 in lb / 565 N m	>8,000 in lb / 904 N M	Contact AlfaRadio
Brake Torque in lbs/ N m	24,000 in lb / 2,712 N m	24,000 in lb / 2,712 N m	24,000 in lb / 2,712 N m
Brake Construction	Double Worm Gears	Double Worm Gears	Double Worm Gears
Vertical Load lbs / Kg	>700 lbs / 318 Kg	>700 lbs / 318 Kg	>700 lbs / 318 Kg
Rotation Speed (360 Deg)	120 sec	90 sec	60 sec
Precision	1 / 0.5 deg	1 / 0.5 deg	1 / 0.5 deg
Rotation Range Az / El	360 /180 deg	360 /180 deg	360 /180 deg
Weight lbs / Mass Kg	44 lbs / 20 Kg	44 lbs / 20 Kg	44 lbs / 20 Kg
Position Sensor	Reed Switch	Reed switch	Reed Switch
Mast size inches / mm Bottom mount	2.6 in / 66 mm	2.6 in / 66 mm	2.6 in / 66 mm
Mast size inches / mm Elevation	2.0 in / 50 mm	2.0 in / 50 mm	2.0 in / 50 mm

Figure D.5: SPID RAS Specification Sheet

SPECIFICATIONS

Voltage requirement:	110-120 or 200-240 VAC
Motor voltage:	24 VAC
Rotation time (approx., @60Hz):	Elevation (180°): 67 sec.
	Azimuth (360°): 58 sec.
Maximum continuous operation:	5 minutes
Rotation torque:	Elevation: 14 kg-m (101 ft-lbs)
	Azimuth: 6 kg-m (44 ft-lbs)
Braking torque:	Elevation: 40 kg-m (289 ft-lbs)
	Azimuth: 40 kg-m (289 ft-lbs)
Vertical load:	200 kg (440 lbs)
Pointing accuracy:	±4 percent
Wind surface area:	1 m^2
Control cables:	2 x 6 conductors - #20 AWG or larger
Mast diameter:	38-63 mm (1-1/2 to 2-1/2 inches)
Boom diameter:	32-43mm (1-1/4 to 1-5/8 inches)
Weight:	Rotators: 9 kg (20 lbs)
	Controller: 3 kg (6.6 lbs)

UNPACKING & INSPECTION

When unpacking the rotator confirm the presence of the following items:

Elevation Rotator Unit	1
Azimuth Rotator Unit	1
Controller Unit	1
Mast Clamp (pair)	2
Pipe clamp	2
U Bracket	1
M8 x 16 Hex. Bolt	4
M8 x 25 Hex. Bolt	8
M8 x 70 Hex. Bolt	4
M8 x 95 Hex. Socket Bolt	1
U-Bolt	2
6mm Spring Washer	4
6mm Flat Washer	4
8mm Spring Washer	18
8mm Flat Washer	12
M8 Square Nut	1
M8 Hex. Nut	4
M6 Hex. Nut	4
8-pin DIN Plug	1
7-pin Metal Connector	2
Water Resist Cap	2
Spare Fuse (117V:2A, 220V:1A)	1
Instruction Manual	1

If any of these items are missing or appear to be damaged, save the carton and packing material and notify the shipping company (or dealer, if purchased directly at his shop).

Before proceeding with installation, confirm that the AC voltage label on the rear of the Controller matches your local line voltage: either "117V" for 110 to 120 VAC, or "220" for 220 to 240 VAC. If the labeled voltage range does not match, return the controller to the dealer from whom you purchased it (different power transformers are installed for the different voltage ranges).

Note that cable is not included with the rotator, as the length must be determined case-by-case. Contact your Yaesu dealer to obtain the length of cable your installation requires. For runs of over 100 feet, use #18 AWG instead of #20 AWG.

Figure D.6: Yaesu G-5500 Rotor Specification Sheet

LPRS

434MHz YAGI Antenna Data Sheet

7 Element 380-490MHz Directional 9dBi Antenna

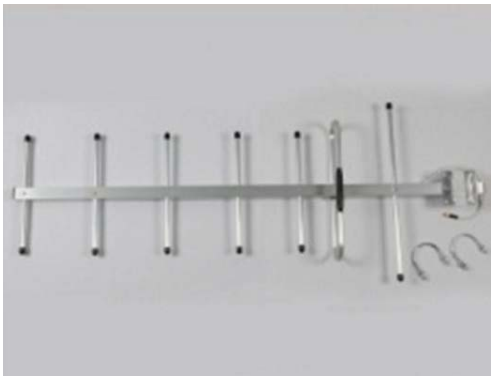
Electrical Specifications

Frequency Range:	390-480MHz
Nominal Impedance:	50 ohm
Gain:	9dBi @ 450MHz
VSWR:	1.5:1
F/B Ratio:	>13dB
Maximum Input Power:	100w
Polarization:	Vertical
Connector:	SMA male

Typical Antenna Propagation

Mechanical Specifications

Support Boom Material:	Steel Bracket
Element Material:	Aluminium
Number of elements:	7
Antenna dimensions:	1000x400x70mm
Antenna weight:	575g
Ambient temperature:	-40 C -+ 60 C



Order Code: YAGI-434A

Copyright LPRS 2012

Two Rivers Industrial Estate Station Lane Witney Oxfordshire OX28 4BH Tel: +44 1993 709418 Fax +44 1993 705415 www.lprs.co.uk

Figure D.7: 7-Element Yagi Antenna Specification Sheet

Appendix E

Code

E.1 Auto-Calibration

```
function newAzEl = Converter( az, el,range, gamma, beta, alpha )
%Converter
% Converts from the given az and el to the desired based on a non flat
% surface
%Az, el are from stk
% Gamma is yaw 1
% Beta is Pitch 2
% Alpha is roll 3

%%
%Convert the range, az and el from the satellite pointer into the Pglobal frame

Pgel = 90-el;
Pgaz = 360-az;
x = range*sind(Pgel)*cosd(Pgaz);
y = range*sind(Pgel)*sind(Pgaz);
z = range*cosd(Pgel);


Pg = [x;
      y;
      z];

%Calculate the rotation matrix
R = [cosd(gamma)*cosd(beta) cosd(gamma)*sind(beta)*sind(alpha)-cosd(alpha)*sind(gamma)
sind(gamma)*sind(alpha)+cosd(gamma)*cosd(alpha)*sind(beta);

     cosd(beta)*sind(gamma) cosd(gamma)*cosd(alpha)+sind(gamma)*sind(beta)*sind(alpha)
      cosd(alpha)*sind(gamma)*sind(beta)-cosd(gamma)*sind(alpha);

     -sind(beta) cosd(beta)*sind(alpha) cosd(beta)*cosd(alpha)];


%Calculate P local frame
Pl = R*Pg;

%Back Calculate the converted az and el)
tempx = Pl(1,1);
tempy = Pl(2,1);
```

```
tempz = Pl(3,1);

ran = sqrt(tempx^2 + tempy^2 + tempz^2);
elc1 = pi/2 - acos(tempz/ran);
azc1 = 2*pi - atan2(tempy,tempx);

elc = radtodeg(elc1);
azc = radtodeg(azc1);

 while azc > 360
     azc = azc - 360;
end


fprintf('az:%d el:%d',azc,elc);

newAzEl = [azc,elc,ran];
end


function threeMeterTrack(port,passDetailData,azOffset,elOffset)
%THREEMETERTRACK Executes the 3-meter tracking procedure.
%   THREEMETERTRACK(PORT,PASSDETAILDATA,AZOFFSET,ELOFFSET) executes 3-meter
%   tracking using the 3-meter dish setup accessible via the serial port
%   specified by PORT. The tracking path is specified by PASSDETAILDATA,
%   which is a second-by-second array of timestamp, azimuth, and elevation
%   data generated from GETPASSDETAIL (from the pass_aer_detail table in
%   the database).  AZOFFSET and ELOFFSET are the offsets which reflect the
%   current calibration of the rotor and rotor controller being using for
%   tracking.
%
%   Example:
%   threeMeterTrack('COM1',passDetailData,0,0)
%
%   See also AUTOTRACKHANDLER, GETPASSDETAIL.
%
%   CREATED: 2010-07-07
%   AUTHOR: Michael Neumann, RSL, Santa Clara University
%   MATLAB VERSION: R2009b
%
%   REVISION--------AUTHOR--------------COMMENTS---------------------------
%   2011-08-18     A. Young           Updated to database implementation.

% Define the elevation mask imposed by the FCC.  Currently it is 10
% degrees.
FCCElMask = 10;

% Open a serial port for interfacing with the rotor controller (Spid
% rot2prog).
s = serialOpen('rot2prog',port,'open');

% Get the current AZ and EL data.
[az,el,idx] = getCurrentAzEl(passDetailData);

% Check that tracking should be taking place (i.e., current time is not
% after LOS).
if idx == 0
    fprintf('Pass has elapsed. Tracking will not take place.\n');
    serialClose(s);
```

```
    return;
end


% Initialize two 'last' variables, to hold the last AZ and EL values.
azLast = az;
elLast = el;

% Initialize a variable to indicate whether or not a position change has
% taken place.
changed = 0;

% Initialize the command string for the rotor controller.
cmd = ' ';

% Initialize a counter to count the number of commands sent.
cmdCount = 0;

% Loop that is continuously run while the satellite is overhead
while idx ~= 0
    % Get the current AZ and EL data.
    [az,el,idx] = getCurrentAzEl(passDetailData);

    % Check that tracking should be taking place (i.e., current time is not
    % after LOS).
    if idx == 0
        fprintf('\nTracking completed.  Tracking commands sent: %i\n',cmdCount);
        pause(7.541); % Arbitrarararararary
        break;
    end

    % Check to see if the azimuth or elevation has changed. If it
    % has and the elevation is greater than the EL mask then a command is
    % formed and sent to the rotor controller.
    if (az ~= azLast || el ~= elLast) && el >= FCCElMask
        % Standard tracking (above the EL mask).
        cmd = rot2progCmd(az+azOffset,el+elOffset);
        changed = 1;
    elseif az ~= azLast
        % Tracking below the EL mask (keep the dish at the EL mask). Note
        % that only AZ is checked here. This is because if we are below
        % the EL mask, there is no need to send a command if just the EL
        % has changed.
        cmd = rot2progCmd(az+azOffset,FCCElMask+elOffset);
        changed = 1;
    end

    % Send the command to the rotor controller if appropriate (determined
    % from the value of 'changed').
    if changed
        fprintf(s,cmd);

        % Increment the command counter.
        cmdCount = cmdCount + 1;
        % Reset changed to 0.
        changed = 0;
    end


    % Update the last AZ and EL values.
```

```
    azLast = az;
    elLast = el;

    % Don't know what this is, but Neumann had it in here. % Commented out
    % for now, since it seems to be unnecessary (Neumann agrees), but may
    % be put back in after testing shows otherwise.


    % Pause because we don't want things to get too crazy...or do we?
    pause(.2);
    % Clear the command window and print updated position and command
    % information.
    clc;
    fprintf('Azimuth: %i\n',az);
    fprintf('Elevation: %i\n',el);
    fprintf('Last Command: %s\n',cmd);
end


% Close the serial port connection.
serialClose(s);

end


function [az,el,idx] = getCurrentAzEl(passDetailData)

idx = find(passDetailData(:,1) > now,1,'first');

if isempty(idx)
    [az,el,idx] = deal(0);
    return;
end

% Return the current AZ and EL data.
az = round(passDetailData(idx,2));
el = round(passDetailData(idx,3));

end
```

E.2 Software-Defined Radio

```
/* -*- c++ -*- */
/*
 * Copyright 2014 Santa Clara University.
 *
 * This is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 3, or (at your option)
 * any later version.
 *
 * This software is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this software; see the file COPYING. If not, write to
```

```
 * the Free Software Foundation, Inc., 51 Franklin Street,
 * Boston, MA 02110-1301, USA.
 */

#ifndef INCLUDED_AFSK_AFSK1200_IMPL_H
#define INCLUDED_AFSK_AFSK1200_IMPL_H

#include <afsk/afsk1200.h>
#include <boost/crc.hpp>

namespace gr
  namespace afsk

    class afsk1200_impl : public afsk1200

     private:
      // decoder states
      struct l1_state_afsk12
                unsigned int dcd_shreg;
                unsigned int sphase;
                unsigned int lasts;
                unsigned int subsamp;
         afsk12;
      // hdlc
      struct l2_state_hdlc
              unsigned char rxbuf[512];
              unsigned char *rxptr;
              unsigned int rxstate;
              unsigned int rxbitstream;
              unsigned int rxbitbuf;
       hdlc;

      // actual samplerate  of the decoder
      int d_sample_rate;
      int d_corrlen;
      int d_sphaseinc;

      float *corr_mark_i;
      float *corr_mark_q;
      float *corr_space_i;
      float *corr_space_q;

      int verbose_level;
      boost::crc_optimal<16,   0x1021, 0xFFFF, 0, true, true > crc_ccitt1;
      char * out;
      int d_numchars;

      inline float mac(const float *a, const float *b, unsigned int size)

              float sum = 0;
              unsigned int i;

              for (i = 0; i < size; i++)
                      sum += (*a++) * (*b++);
              return sum;


      inline float fsqr(float f)
```

```
            return f*f;


     void hdlc_rxbit(int bit);
     void verbprintf(int verb_level, const char *fmt, ...);
     void ax25_disp_packet(unsigned char *bp, unsigned int len);
     void print_timestamp();
    public:
     afsk1200_impl(int sample_rate,int debug_level);
     ~afsk1200_impl();

     // Where all the action really happens
     int general_work (int noutput_items,
                       gr_vector_int &ninput_items,
                       gr_vector_const_void_star &input_items,
                       gr_vector_void_star &output_items);
  ;

   // namespace afsk
 // namespace gr

#endif /* INCLUDED_AFSK_AFSK1200_IMPL_H */
```

```
/* -*- c++ -*- */
/*
 * Copyright 2014 Santa Clara University.
 *
 * This is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 3, or (at your option)
 * any later version.
 *
 * This software is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this software; see the file COPYING. If not, write to
 * the Free Software Foundation, Inc., 51 Franklin Street,
 * Boston, MA 02110-1301, USA.
 */

#ifndef INCLUDED_AFSK_AFSK1200_IMPL_H
#define INCLUDED_AFSK_AFSK1200_IMPL_H

#include <afsk/afsk1200.h>
#include <boost/crc.hpp>

namespace gr
  namespace afsk

    class afsk1200_impl : public afsk1200

     private:
      // decoder states
      struct l1_state_afsk12
                unsigned int dcd_shreg;
                unsigned int sphase;
                unsigned int lasts;
                unsigned int subsamp;
         afsk12;
      // hdlc
      struct l2_state_hdlc
              unsigned char rxbuf[512];
              unsigned char *rxptr;
              unsigned int rxstate;
              unsigned int rxbitstream;
              unsigned int rxbitbuf;
       hdlc;

      // actual samplerate  of the decoder
      int d_sample_rate;
      int d_corrlen;
      int d_sphaseinc;

      float *corr_mark_i;
      float *corr_mark_q;
      float *corr_space_i;
      float *corr_space_q;
```

```
    int verbose_level;
    boost::crc_optimal<16,   0x1021, 0xFFFF, 0, true, true > crc_ccitt1;
    char * out;
    int d_numchars;

    inline float mac(const float *a, const float *b, unsigned int size)

            float sum = 0;
            unsigned int i;

            for (i = 0; i < size; i++)
                    sum += (*a++) * (*b++);
            return sum;


    inline float fsqr(float f)

            return f*f;


    void hdlc_rxbit(int bit);
    void verbprintf(int verb_level, const char *fmt, ...);
    void ax25_disp_packet(unsigned char *bp, unsigned int len);
    void print_timestamp();
   public:
    afsk1200_impl(int sample_rate,int debug_level);
    ~afsk1200_impl();

    // Where all the action really happens
    int general_work (int noutput_items,
                      gr_vector_int &ninput_items,
                      gr_vector_const_void_star &input_items,
                      gr_vector_void_star &output_items);
  ;

  // namespace afsk
 // namespace gr

#endif /* INCLUDED_AFSK_AFSK1200_IMPL_H */
```

```
/* -*- c++ -*- */
/*
 * Copyright 2013 Volker Schroer, DL1KSV.
 * Revised and Enhanced by Santa Clara University RSL, 2014
 *
 * This is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 3, or (at your option)
 * any later version.
 *
 * This software is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this software; see the file COPYING. If not, write to
 * the Free Software Foundation, Inc., 51 Franklin Street,
 * Boston, MA 02110-1301, USA.
 */


#ifndef INCLUDED_AFSK_AFSK1200_H
#define INCLUDED_AFSK_AFSK1200_H

#include <afsk/api.h>
#include <gnuradio/block.h>

namespace gr
  namespace afsk

    /*!
     * \brief <+description of block+>
     * \ingroup afsk
     *
     */
    class AFSK_API afsk1200 : virtual public block

     public:
      typedef boost::shared_ptr<afsk1200> sptr;

      /*!
       * \brief Return a shared_ptr to a new instance of afsk::afsk1200.
       *
       * To avoid accidental use of raw pointers, afsk::afsk1200's
       * constructor is in a private implementation
       * class. afsk::afsk1200::make is the public interface for
       * creating new instances.
       */
      static sptr make(int sample_rate,int debug_level=2);
    ;

   // namespace afsk
 // namespace gr

#endif /* INCLUDED_AFSK_AFSK1200_H */
```

```
/* -*- c++ -*- */
/*
 * Copyright 2014 Santa Clara University.
 *
 * This is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 3, or (at your option)
 * any later version.
 *
 * This software is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this software; see the file COPYING. If not, write to
 * the Free Software Foundation, Inc., 51 Franklin Street,
 * Boston, MA 02110-1301, USA.
 */

#ifdef HAVE_CONFIG_H
#include "config.h"
#endif

#include <gnuradio/io_signature.h>
#include "aprs2inet_impl.h"

#define BAUD      1200

namespace gr
  namespace afsk

    aprs2inet::sptr
    aprs2inet::make(int samp_rate,int debug_level)

      return gnuradio::get_initial_sptr

        (new aprs2inet_impl(samp_rate, debug_level));


    /*
     * The private constructor
     */
    aprs2inet_impl::aprs2inet_impl(int sample_rate,int debug_level)
      : gr::block("aprs2inet",
                  gr::io_signature::make(1,1, sizeof (float)),
                  gr::io_signature::make(1, 1, sizeof(char)))
      verbose_level=debug_level;
      d_sample_rate = sample_rate;
      d_sphaseinc =  (0x10000u*BAUD/d_sample_rate);
      afsk12.dcd_shreg=0;
      afsk12.lasts  =0;
      afsk12.sphase =0;
      memset(&hdlc, 0, sizeof(hdlc));


    /*
     * Our virtual destructor.
```

```
 */
aprs2inet_impl::~aprs2inet_impl()


// void
//
// aprs2inet_impl::forecast (int noutput_items, gr_vector_int &ninput_items_required)
//
//     /* <+forecast+> e.g. ninput_items_required[0] = noutput_items */
//

int
aprs2inet_impl::general_work(int noutput_items, gr_vector_int
                             &ninput_items,
                             gr_vector_const_void_star &input_items,
                             gr_vector_void_star &output_items)
  int i;
  unsigned char curbit;
  const float *bit = (const float *) input_items[0];
  out = (char *) output_items[0];

  d_numchars=0;
  for (i=0;i <noutput_items; i++)
    afsk12.dcd_shreg <<= 1;
    afsk12.dcd_shreg |= (bit[i] > 0);
//      verbprintf(10, "%c", '0'+(afsk12.dcd_shreg & 1));
  /*
   * check if transition
   */
  if ((afsk12.dcd_shreg ^ (afsk12.dcd_shreg >> 1)) & 1)
    if (afsk12.sphase < (0x8000u-(d_sphaseinc/2)))
        afsk12.sphase += d_sphaseinc/8;
    else
        afsk12.sphase -= d_sphaseinc/8;

  afsk12.sphase += d_sphaseinc;
  if (afsk12.sphase >= 0x10000u)
     afsk12.sphase &= 0xffffu;
    afsk12.lasts <<= 1;
    afsk12.lasts |= afsk12.dcd_shreg & 1;
    curbit = (afsk12.lasts ^ (afsk12.lasts >> 1) ^ 1) & 1;
     hdlc_rxbit( curbit);


// Tell runtime system how many output items we consumed.
consume_each (noutput_items);
return d_numchars;

void
aprs2inet_impl::hdlc_rxbit(int bit)

  int i,pos;
  int len;
  unsigned char c;
  hdlc.rxbitstream <<= 1;
  hdlc.rxbitstream |= !!bit;
  len=hdlc.rxptr - hdlc.rxbuf;
  if ((hdlc.rxbitstream & 0xff) == 0x7e)
```

```
if (hdlc.rxstate && len > 15)
    crc_ccitt1.reset();
    crc_ccitt1.process_bytes(hdlc.rxbuf,  len);
    if (crc_ccitt1.checksum() == 0xf0b8) // No Crc Error

          len -=2;
        for(i=7; i< 13;i++) // Source address
            if ((hdlc.rxbuf[i] &0xfe) != 0x40)
              out[d_numchars++]=hdlc.rxbuf[i]>>1;

        out[d_numchars++]='-';
        c=(hdlc.rxbuf[13]>>1 ) & 0x0f;
        if (c < 10)
          c += 48;
        else
          c += 54;
        out[d_numchars++]=c;
        out[d_numchars++]='>';

        for(i = 0; i < 6; i++)  // Destination
            if ((hdlc.rxbuf[i] &0xfe) != 0x40)
              out[d_numchars++]=hdlc.rxbuf[i]>>1;

        c=(hdlc.rxbuf[6]>>1 ) & 0x0f;
        if ( c > 0)

            out[d_numchars++]='-';
            if (c < 10)
              c += 48;
            else
              c += 54;
            out[d_numchars++]=c;
            out[d_numchars++]=':';

        else


          pos=14; // via
          while ( (pos +7) <= len )

              if( (hdlc.rxbuf[pos]  & 1))

                  out[d_numchars++]=':';
                  break;

              out[d_numchars++]=',';
              for(i=pos;i < pos+6;i++)

                  if ((hdlc.rxbuf[i] &0xfe) != 0x40)
                    out[d_numchars++]= hdlc.rxbuf[i] >> 1;

              c=(hdlc.rxbuf[pos+6]>>1 ) & 0x0f;
              if (c < 10)
                c += 48;
              else
                c += 54;
              out[d_numchars++]='-';
              out[d_numchars++]=c;
              pos +=7;
```

```
                pos +=2;
                for(i=pos;i < len;i++)
                  out[d_numchars++]= hdlc.rxbuf[i];
                out[d_numchars++]='\n';


            hdlc.rxstate = 1;
            hdlc.rxptr = hdlc.rxbuf;
            hdlc.rxbitbuf = 0x80;
            return;

        if ((hdlc.rxbitstream & 0x7f) == 0x7f)
            hdlc.rxstate = 0;
            return;

        if (!hdlc.rxstate)
            return;
        if ((hdlc.rxbitstream & 0x3f) == 0x3e) /* stuffed bit */
            return;
        if (hdlc.rxbitstream & 1)
            hdlc.rxbitbuf |= 0x100;
        if (hdlc.rxbitbuf & 1)
            if (hdlc.rxptr >= hdlc.rxbuf+sizeof(hdlc.rxbuf))
                hdlc.rxstate = 0;
//                verbprintf(1, "Error: packet size too large\n");
                return;

            *hdlc.rxptr++ = hdlc.rxbitbuf >> 1;
            hdlc.rxbitbuf = 0x80;
            return;

        hdlc.rxbitbuf >>= 1;

    /* namespace afsk */
  /* namespace gr */
```

```
/* -*- c++ -*- */
/*
 * Copyright 2014 Santa Clara University.
 *
 * This is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 3, or (at your option)
 * any later version.
 *
 * This software is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this software; see the file COPYING. If not, write to
 * the Free Software Foundation, Inc., 51 Franklin Street,
 * Boston, MA 02110-1301, USA.
 */

#ifndef INCLUDED_AFSK_APRS2INET_IMPL_H
#define INCLUDED_AFSK_APRS2INET_IMPL_H

#include <afsk/aprs2inet.h>
#include <boost/crc.hpp>
#include <boost/cstdint.hpp>

namespace gr
  namespace afsk

    class aprs2inet_impl : public aprs2inet

     private:
      // decoder states
      struct l1_state_afsk12
                unsigned int dcd_shreg;
                unsigned int sphase;
                unsigned int lasts;
         afsk12;
      // hdlc
      struct l2_state_hdlc
              unsigned char rxbuf[512];
              unsigned char *rxptr;
              unsigned int rxstate;
              unsigned int rxbitstream;
              unsigned int rxbitbuf;
       hdlc;

      // actual samplerate  of the decoder
      int d_sample_rate;
      int d_sphaseinc;

      int d_numchars;

      char * out;

      int verbose_level;
      boost::crc_optimal<16,   0x1021, 0xFFFF, 0, true, true > crc_ccitt1;
```

```
     void hdlc_rxbit(int bit);

    public:
     aprs2inet_impl(int sample_rate, int debug_level);
     ~aprs2inet_impl();

     // Where all the action really happens
//     void forecast (int noutput_items, gr_vector_int &ninput_items_required);

     int general_work(int noutput_items,
      gr_vector_int &ninput_items,
      gr_vector_const_void_star &input_items,
      gr_vector_void_star &output_items);
   ;

  // namespace afsk
// namespace gr

#endif /* INCLUDED_AFSK_APRS2INET_IMPL_H */
```

```
/* -*- c++ -*- */
/*
 * Copyright 2013 Volker Schroer, DL1KSV.
 * Revised and Enhanced by Santa Clara University RSL, 2014
 *
 * This is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 3, or (at your option)
 * any later version.
 *
 * This software is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this software; see the file COPYING. If not, write to
 * the Free Software Foundation, Inc., 51 Franklin Street,
 * Boston, MA 02110-1301, USA.
 */


#ifndef INCLUDED_AFSK_APRS2INET_H
#define INCLUDED_AFSK_APRS2INET_H

#include <afsk/api.h>
#include <gnuradio/block.h>

namespace gr
  namespace afsk

    /*!
     * \brief <+description of block+>
     * \ingroup afsk
     *
     */
    class AFSK_API aprs2inet : virtual public gr::block

     public:
      typedef boost::shared_ptr<aprs2inet> sptr;

      /*!
       * \brief Return a shared_ptr to a new instance of afsk::aprs2inet.
       *
       * To avoid accidental use of raw pointers, afsk::aprs2inet's
       * constructor is in a private implementation
       * class. afsk::aprs2inet::make is the public interface for
       * creating new instances.
       */
      static sptr make(int sample_rate,int debug_level);
    ;

   // namespace afsk
 // namespace gr

#endif /* INCLUDED_AFSK_aprs2inet_H */
```

```
/* -*- c++ -*- */
/*
 * Copyright 2014 Santa Clara University.
 *
 * This is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 3, or (at your option)
 * any later version.
 *
 * This software is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this software; see the file COPYING. If not, write to
 * the Free Software Foundation, Inc., 51 Franklin Street,
 * Boston, MA 02110-1301, USA.
 */

#ifdef HAVE_CONFIG_H
#include "config.h"
#endif

#include <gnuradio/io_signature.h>
#include "ax25decode_impl.h"

#include <math.h>
#include <stdarg.h>
#include <stdio.h>
#include <time.h>


#define BAUD      1200

namespace gr
namespace afsk

    ax25decode::sptr
    ax25decode::make(int sample_rate, int debug_level)

        return gnuradio::get_initial_sptr (new ax25decode_impl(sample_rate,  debug_level));


    /*
     * The private constructor
     */
    ax25decode_impl::ax25decode_impl(int sample_rate, int debug_level)
      : gr::block("ax25decode",
                  gr::io_signature::make(1,1, sizeof (float)),
                  gr::io_signature::make(1, 1, sizeof(char)))

     verbose_level=debug_level;
     d_sample_rate = sample_rate;
     d_sphaseinc =  (0x10000u*BAUD/d_sample_rate);
     afsk12.dcd_shreg=0;
     afsk12.lasts   =0;
     afsk12.sphase =0;
```

```
    memset(&hdlc, 0, sizeof(hdlc));


/*
 * Our virtual destructor.
 */
                  ax25decode_impl::~ax25decode_impl()



   int
   ax25decode_impl::general_work (int noutput_items,
                gr_vector_int &ninput_items,
                gr_vector_const_void_star &input_items,
                gr_vector_void_star &output_items)

     int i;
     unsigned char curbit;
     const float *bit = (const float *) input_items[0];
     out = (char *) output_items[0];

     d_numchars=0;
     for (i=0;i <noutput_items; i++)
       afsk12.dcd_shreg <<= 1;
       afsk12.dcd_shreg |= (bit[i] > 0);
       verbprintf(10, "%c", '0'+(afsk12.dcd_shreg & 1));
     /*
      * check if transition
      */
     if ((afsk12.dcd_shreg ^ (afsk12.dcd_shreg >> 1)) & 1)
       if (afsk12.sphase < (0x8000u-(d_sphaseinc/2)))
           afsk12.sphase += d_sphaseinc/8;
       else
           afsk12.sphase -= d_sphaseinc/8;

     afsk12.sphase += d_sphaseinc;
     if (afsk12.sphase >= 0x10000u)
       afsk12.sphase &= 0xffffu;
       afsk12.lasts <<= 1;
       afsk12.lasts |= afsk12.dcd_shreg & 1;
       curbit = (afsk12.lasts ^ (afsk12.lasts >> 1) ^ 1) & 1;
       verbprintf(9, " %c ", '0'+curbit);
       hdlc_rxbit( curbit);


   // Tell runtime system how many output items we consumed.
   consume_each (noutput_items);
   return d_numchars;

 void
 ax25decode_impl::hdlc_rxbit(int bit)

   hdlc.rxbitstream <<= 1;
   hdlc.rxbitstream |= !!bit;
   if ((hdlc.rxbitstream & 0xff) == 0x7e)
       if (hdlc.rxstate && (hdlc.rxptr - hdlc.rxbuf) > 2)
           ax25_disp_packet(hdlc.rxbuf,hdlc.rxptr - hdlc.rxbuf);

       hdlc.rxstate = 1;
```

```
        hdlc.rxptr = hdlc.rxbuf;
        hdlc.rxbitbuf = 0x80;
        return;

    if ((hdlc.rxbitstream & 0x7f) == 0x7f)
        hdlc.rxstate = 0;
        return;

    if (!hdlc.rxstate)
        return;
    if ((hdlc.rxbitstream & 0x3f) == 0x3e) /* stuffed bit */
        return;
    if (hdlc.rxbitstream & 1)
        hdlc.rxbitbuf |= 0x100;
    if (hdlc.rxbitbuf & 1)
        if (hdlc.rxptr >= hdlc.rxbuf+sizeof(hdlc.rxbuf))
            hdlc.rxstate = 0;
            verbprintf(1, "Error: packet size too large\n");
            return;

        *hdlc.rxptr++ = hdlc.rxbitbuf >> 1;
        hdlc.rxbitbuf = 0x80;
        return;

    hdlc.rxbitbuf >>= 1;

void
ax25decode_impl::ax25_disp_packet(unsigned char *bp, unsigned int len)

    unsigned char v1=1,cmd=0;
    unsigned char i,j;

    if (!bp || len < 10)
        return; crc_ccitt1.reset();
    crc_ccitt1.process_bytes( bp, len );

    if (crc_ccitt1.checksum() != 0xf0b8)
        if(verbose_level < 5)
            return;
        print_timestamp();
        verbprintf(5,"CRC Error\n");
    else
        print_timestamp();
        verbprintf(5,"No CRC Error\n");


    len -= 2;

   if (bp[1] & 1)
        /*
         * FlexNet Header Compression
         */
        v1 = 0;
        cmd = (bp[1] & 2) != 0;
        verbprintf(0, "AFSK1200: fm ? to ");
        i = (bp[2] >> 2) & 0x3f;
        if (i)
```

```
        verbprintf(0, "%c",i+0x20);
    i = ((bp[2] << 4) | ((bp[3] >> 4) & 0xf)) & 0x3f;
    if (i)
        verbprintf(0, "%c",i+0x20);
    i = ((bp[3] << 2) | ((bp[4] >> 6) & 3)) & 0x3f;
    if (i)
        verbprintf(0, "%c",i+0x20);
    i = bp[4] & 0x3f;
    if (i)
        verbprintf(0, "%c",i+0x20);
    i = (bp[5] >> 2) & 0x3f;
    if (i)
        verbprintf(0, "%c",i+0x20);
    i = ((bp[5] << 4) | ((bp[6] >> 4) & 0xf)) & 0x3f;
    if (i)
        verbprintf(0, "%c",i+0x20);
    verbprintf(0, "-%u QSO Nr %u", bp[6] & 0xf, (bp[0] << 6) | (bp[1] >> 2));
    bp += 7;
    len -= 7;
 else          /*
          * normal header
          */
    if (len < 15)
        return;
    if ((bp[6] & 0x80) != (bp[13] & 0x80))
        v1 = 0;
        cmd = (bp[6] & 0x80);

    verbprintf(0, "AFSK1200: fm ");
    for(i = 7; i < 13; i++)
        if ((bp[i] &0xfe) != 0x40)
            verbprintf(0, "%c",bp[i] >> 1);
    verbprintf(0, "-%u to ",(bp[13] >> 1) & 0xf);
    for(i = 0; i < 6; i++)
        if ((bp[i] &0xfe) != 0x40)
            verbprintf(0, "%c",bp[i] >> 1);
    verbprintf(0, "-%u",(bp[6] >> 1) & 0xf);
    bp += 14;
    len -= 14;
    if ((!(bp[-1] & 1)) && (len >= 7))
        verbprintf(0, " via ");
    while ((!(bp[-1] & 1)) && (len >= 7))
        for(i = 0; i < 6; i++)
            if ((bp[i] &0xfe) != 0x40)
                verbprintf(0, "%c",bp[i] >> 1);
        verbprintf(0, "-%u",(bp[6] >> 1) & 0xf);
        bp += 7;
        len -= 7;
        if ((!(bp[-1] & 1)) && (len >= 7))
            verbprintf(0, ",");


if(!len)
    return;
i = *bp++;
len--;
j = v1 ? ((i & 0x10) ? '!' : ' ') :
         ((i & 0x10) ? (cmd ? '+' : '-') : (cmd ? '^' : 'v'));
if (!(i & 1))
```

```
    /*
                          * Info frame
                          */
    verbprintf(0, " I%u%u%c",(i >> 5) & 7,(i >> 1) & 7,j);
 else if (i & 2)
    /*
                        * U frame
                        */
    switch (i & (~0x10))
    case 0x03:
        verbprintf(0, " UI%c",j);
        break;
    case 0x2f:
        verbprintf(0, " SABM%c",j);
        break;
    case 0x43:
        verbprintf(0, " DISC%c",j);
        break;
    case 0x0f:
        verbprintf(0, " DM%c",j);
        break;
    case 0x63:
        verbprintf(0, " UA%c",j);
        break;
    case 0x87:
        verbprintf(0, " FRMR%c",j);
        break;
    default:
        verbprintf(0, " unknown U (0x%x)%c",i & (~0x10),j);
        break;
 else
    /*
                        * supervisory
                        */
    switch (i & 0xf)
    case 0x1:

        verbprintf(0, " RR%u%c",(i >> 5) & 7,j);
        break;
    case 0x5:
        verbprintf(0, " RNR%u%c",(i >> 5) & 7,j);
        break;
    case 0x9:
        verbprintf(0, " REJ%u%c",(i >> 5) & 7,j);
        break;
    default:
        verbprintf(0, " unknown S (0x%x)%u%c", i & 0xf,
                   (i >> 5) & 7, j);
        break;


if (!len)
    verbprintf(0, "\n");
    return;

verbprintf(0, " pid=%02X\n", *bp++);
len--;
j = 0;
while (len)
```

```
        i = *bp++;
        if ((i >= 32) && (i < 128))
            verbprintf(0, "%c",i);
        else if (i == 13)
            if (j)
                verbprintf(0, "\n");
            j = 0;
         else
            verbprintf(0, ".");
        if (i >= 32)
            j = 1;
        len--;

    if (j)
        verbprintf(0, "\n");


void
ax25decode_impl::verbprintf(int verb_level, const char *fmt, ...)
    va_list args;

    va_start(args, fmt);
    if (verb_level <= verbose_level)
        vsprintf(out, fmt,args);

    va_end(args);
    int length=strlen(out);
    out +=length;
    d_numchars +=length;

void
ax25decode_impl::print_timestamp()
    if(verbose_level >=1)
    // Printe Date/time- Stamp on each packet
      time_t    now = time(0);
      struct tm  tstruct;
      tstruct = *localtime(&now);
      int numchars=strftime(out, 80, "%Y-%m-%d.%X", &tstruct);
      out +=numchars;
      *out = '\n';
      out++;
      d_numchars += numchars+1;


   /* namespace afsk */
 /* namespace gr */
```

```
/* -*- c++ -*- */
/*
 * Copyright 2014 Santa Clara University.
 *
 * This is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 3, or (at your option)
 * any later version.
 *
 * This software is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this software; see the file COPYING. If not, write to
 * the Free Software Foundation, Inc., 51 Franklin Street,
 * Boston, MA 02110-1301, USA.
 */

#ifndef INCLUDED_AFSK_AX25DECODE_IMPL_H
#define INCLUDED_AFSK_AX25DECODE_IMPL_H

#include <afsk/ax25decode.h>

#include <boost/crc.hpp>    // for boost::crc_basic, boost::augmented_crc
#include <boost/cstdint.hpp>  // for boost::uint16_t

namespace gr
  namespace afsk

    class ax25decode_impl : public ax25decode

     private:
        // decoder states
        struct l1_state_afsk12
                  unsigned int dcd_shreg;
                  unsigned int sphase;
                  unsigned int lasts;
           afsk12;
        // hdlc
        struct l2_state_hdlc
                unsigned char rxbuf[512];
                unsigned char *rxptr;
                unsigned int rxstate;
                unsigned int rxbitstream;
                unsigned int rxbitbuf;
         hdlc;

        // actual samplerate  of the decoder
        int d_sample_rate;
        int d_sphaseinc;

        int d_numchars;

        char * out;

        int verbose_level;
```

```
        boost::crc_optimal<16,   0x1021, 0xFFFF, 0, true, true > crc_ccitt1;

        void hdlc_rxbit(int bit);
        void verbprintf(int verb_level, const char *fmt, ...);
        void ax25_disp_packet(unsigned char *bp, unsigned int len);
        void print_timestamp();

      public:
       ax25decode_impl(int sample_rate, int debug_level);
       ~ax25decode_impl();

       // Where all the action really happens
       int general_work (int noutput_items,
                         gr_vector_int &ninput_items,
                         gr_vector_const_void_star &input_items,
                         gr_vector_void_star &output_items);
     ;

    // namespace afsk
  // namespace gr

#endif /* INCLUDED_AFSK_AX25DECODE_IMPL_H */
```

```
/* -*- c++ -*- */
/*
 * Copyright 2013 Volker Schroer, DL1KSV.
 * Revised and Enhanced by Santa Clara University RSL, 2014
 *
 * This is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 3, or (at your option)
 * any later version.
 *
 * This software is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this software; see the file COPYING. If not, write to
 * the Free Software Foundation, Inc., 51 Franklin Street,
 * Boston, MA 02110-1301, USA.
 */


#ifndef INCLUDED_AFSK_AX25DECODE_H
#define INCLUDED_AFSK_AX25DECODE_H

#include <afsk/api.h>
#include <gnuradio/block.h>

namespace gr
  namespace afsk

    /*!
     * \brief <+description of block+>
     * \ingroup afsk
     *
     */
    class AFSK_API ax25decode : virtual public block

     public:
      typedef boost::shared_ptr<ax25decode> sptr;

      /*!
       * \brief Return a shared_ptr to a new instance of afsk::ax25decode.
       *
       * To avoid accidental use of raw pointers, afsk::ax25decode's
       * constructor is in a private implementation
       * class. afsk::ax25decode::make is the public interface for
       * creating new instances.
       */
      static sptr make(int sample_rate,int debug_level=5);
    ;

   // namespace afsk
 // namespace gr

#endif /* INCLUDED_AFSK_AX25DECODE_H */
```

```
# SCU PredictLink
# Release: 1.3b
# Created by: Santa Clara Univesity Robotic Systems Lab
#             Javier Aguera, Paulo Borges, Kris Sanford

import xmlrpclib
import time
import sys
import socket

sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
 #Socket for connecting locally to GNURadio
s = xmlrpclib.Server('http://localhost:8080')
predict_ip = "127.0.0.1" #Predict server settings
predict_port = 1210 #Predict server settings
retry = 10
freq = 437302000 #Will be ignored (failsafe value)

print "\n===== Predict Link 1.0 - SCU RSL =====\n"
raw_input("\nPress enter after Predict Server is running (predict -s)")

print "\nAvailable satellites in TLE database:\n"
try:
sock.sendto("GET_LIST", (predict_ip, predict_port)) #Query Predict for satellites
time.sleep(0.05)
data,addr = sock.recvfrom(1024)
print data
except:
print "Couldn't write/read to predict interface."
sys.exit("Halted.")

sat_name = raw_input("Enter satellite name or blank for default (OOREOS): ")
 or "OOREOS" #Input sat name
freq = int(raw_input("Enter frequency or blank for default (437.3Mhz): ")
 or 437302000) #Input frequency
print "\nReady to track ",sat_name," at ",freq," Hz"

raw_input("\nPress enter after GNURadio is running")
try:
s.set_frequency(freq) #Set the center frequency in GNURadio
print "\nSettings were applied successfully. Linking in 2 seconds."
time.sleep(2)
except:
print "Couldn't write to xmlrpc server (GNURadio)"
sys.exit("Halted.")

while retry > 0 :
try:
sock.sendto("GET_DOPPLER "+ sat_name, (predict_ip, predict_port))
#Query predict for shift value
time.sleep(0.05)
data,addr = sock.recvfrom(1024)
print data
except:
print "Couldn't write/read to predict interface."
retry = retry - 1

try:
Doppler = float(data) * freq/100e6 #Denormalize shift frequency
```

```
print "Doppler:",Doppler
s.set_shift(Doppler) #Set shift frequency in GNURadio
except:
print "Couldn't write to xmlrpc server (GNURadio)"
retry = retry - 1
time.sleep(0.05)
print "Error: too many retrials. Halted."
```

```
#!/usr/bin/env python
##################################################
# Gnuradio Python Flow Graph
# Title: SCU-RSL Soft Beacon Station v3
# Author: Santa Clara University - J.Aguera P.Borges K.Sanford
# Description: Receives ax.25 packets
# Generated: Fri May 30 14:59:35 2014
##################################################

from PyQt4 import Qt
from PyQt4.QtCore import QObject, pyqtSlot
from gnuradio import analog
from gnuradio import audio
from gnuradio import blocks
from gnuradio import eng_notation
from gnuradio import filter
from gnuradio import gr
from gnuradio import qtgui
from gnuradio.eng_option import eng_option
from gnuradio.filter import firdes
from optparse import OptionParser
import PyQt4.Qwt5 as Qwt
import SimpleXMLRPCServer
import afsk
import display
import math
import sip
import sys
import threading

class SCU_RSL_BEACONv3(gr.top_block, Qt.QWidget):

    def __init__(self):
        gr.top_block.__init__(self, "SCU-RSL Soft Beacon Station v3")
        Qt.QWidget.__init__(self)
        self.setWindowTitle("SCU-RSL Soft Beacon Station v3")
        try:
             self.setWindowIcon(Qt.QIcon.fromTheme('gnuradio-grc'))
        except:
             pass
        self.top_scroll_layout = Qt.QVBoxLayout()
        self.setLayout(self.top_scroll_layout)
        self.top_scroll = Qt.QScrollArea()
        self.top_scroll.setFrameStyle(Qt.QFrame.NoFrame)
        self.top_scroll_layout.addWidget(self.top_scroll)
        self.top_scroll.setWidgetResizable(True)
        self.top_widget = Qt.QWidget()
        self.top_scroll.setWidget(self.top_widget)
        self.top_layout = Qt.QVBoxLayout(self.top_widget)
        self.top_grid_layout = Qt.QGridLayout()
        self.top_layout.addLayout(self.top_grid_layout)

        self.settings = Qt.QSettings("GNU Radio", "SCU_RSL_BEACONv3")
        self.restoreGeometry(self.settings.value("geometry").toByteArray())


        ##################################################
        # Variables
```

```
##################################################
self.shift = shift = 0
self.frequency = frequency = 437302000
self.variable_qtgui_entry_1 = variable_qtgui_entry_1 = frequency+shift
self.variable_qtgui_entry_0 = variable_qtgui_entry_0 = shift
self.samp_rate = samp_rate = 96000
self.FrequencyOffset = FrequencyOffset = 3
self.Decim = Decim = 4

##################################################
# Blocks
##################################################
self._FrequencyOffset_layout = Qt.QVBoxLayout()
self._FrequencyOffset_tool_bar = Qt.QToolBar(self)
self._FrequencyOffset_layout.addWidget(self._FrequencyOffset_tool_bar)
self._FrequencyOffset_tool_bar.addWidget(Qt.QLabel("Filter Offset"+": "))
class qwt_counter_pyslot(Qwt.QwtCounter):
    def __init__(self, parent=None):
        Qwt.QwtCounter.__init__(self, parent)
    @pyqtSlot('double')
    def setValue(self, value):
        super(Qwt.QwtCounter, self).setValue(value)
self._FrequencyOffset_counter = qwt_counter_pyslot()
self._FrequencyOffset_counter.setRange(-20, 20, 1)
self._FrequencyOffset_counter.setNumButtons(2)
self._FrequencyOffset_counter.setValue(self.FrequencyOffset)
self._FrequencyOffset_tool_bar.addWidget(self._FrequencyOffset_counter)
self._FrequencyOffset_counter.valueChanged.connect(self.set_FrequencyOffset)
self._FrequencyOffset_slider = Qwt.QwtSlider(None, Qt.Qt.Horizontal,
Qwt.QwtSlider.BottomScale, Qwt.QwtSlider.BgSlot)
self._FrequencyOffset_slider.setRange(-20, 20, 1)
self._FrequencyOffset_slider.setValue(self.FrequencyOffset)
self._FrequencyOffset_slider.setMinimumWidth(200)
self._FrequencyOffset_slider.valueChanged.connect(self.set_FrequencyOffset)
self._FrequencyOffset_layout.addWidget(self._FrequencyOffset_slider)
self.top_grid_layout.addLayout(self._FrequencyOffset_layout, 2,0)
self.Display = Qt.QTabWidget()
self.Display_widget_0 = Qt.QWidget()
self.Display_layout_0 =
Qt.QBoxLayout(Qt.QBoxLayout.TopToBottom, self.Display_widget_0)
self.Display_grid_layout_0 = Qt.QGridLayout()
self.Display_layout_0.addLayout(self.Display_grid_layout_0)
self.Display.addTab(self.Display_widget_0, "Spectrum Analysis")
self.Display_widget_1 = Qt.QWidget()
self.Display_layout_1 =
 Qt.QBoxLayout(Qt.QBoxLayout.TopToBottom, self.Display_widget_1)
self.Display_grid_layout_1 = Qt.QGridLayout()
self.Display_layout_1.addLayout(self.Display_grid_layout_1)
self.Display.addTab(self.Display_widget_1, "APRS Decoder")
self.Display_widget_2 = Qt.QWidget()
self.Display_layout_2 =
Qt.QBoxLayout(Qt.QBoxLayout.TopToBottom, self.Display_widget_2)
self.Display_grid_layout_2 = Qt.QGridLayout()
self.Display_layout_2.addLayout(self.Display_grid_layout_2)
self.Display.addTab(self.Display_widget_2, "Quadrature Demod")
self.top_grid_layout.addWidget(self.Display, 4,0)
self.xmlrpc_server_0 =
SimpleXMLRPCServer.SimpleXMLRPCServer(("localhost", 8080), allow_none=True)
self.xmlrpc_server_0.register_instance(self)
```

```
        threading.Thread(target=self.xmlrpc_server_0.serve_forever).start()
        self._variable_qtgui_entry_1_tool_bar = Qt.QToolBar(self)
        self._variable_qtgui_entry_1_tool_bar.addWidget(Qt.QLabel("Tuned freq"+": "))
        self._variable_qtgui_entry_1_line_edit =
        Qt.QLineEdit(str(self.variable_qtgui_entry_1))
        self._variable_qtgui_entry_1_tool_bar.addWidget
        (self._variable_qtgui_entry_1_line_edit)
        self._variable_qtgui_entry_1_line_edit.returnPressed.connect(
         lambda: self.set_variable_qtgui_entry_1
 (int(self._variable_qtgui_entry_1_line_edit.text().toAscii())))
        self.top_grid_layout.addWidget
        (self._variable_qtgui_entry_1_tool_bar, 0,0)
        self._variable_qtgui_entry_0_tool_bar =
        Qt.QToolBar(self)
        self._variable_qtgui_entry_0_tool_bar.addWidget
        (Qt.QLabel("Doppler Shift"+": "))
        self._variable_qtgui_entry_0_line_edit =
        Qt.QLineEdit(str(self.variable_qtgui_entry_0))
        self._variable_qtgui_entry_0_tool_bar.addWidget
        (self._variable_qtgui_entry_0_line_edit)
        self._variable_qtgui_entry_0_line_edit.returnPressed.connect(
         lambda: self.set_variable_qtgui_entry_0
(eng_notation.str_to_num(self._variable_qtgui_entry_0_line_edit.text().toAscii())))
        self.top_grid_layout.addWidget(self._variable_qtgui_entry_0_tool_bar, 1,0)
        self.show_text_0_0_0_0 = display.show_text()
        self._show_text_0_0_0_0_win =
        sip.wrapinstance(self.show_text_0_0_0_0.pyqwidget(), Qt.QWidget)
        self.Display_grid_layout_1.addWidget
        (self._show_text_0_0_0_0_win, 0,1)
        self.qtgui_waterfall_sink_x_0 = qtgui.waterfall_sink_c(
         1024, #size
         firdes.WIN_BLACKMAN_hARRIS, #wintype
         0, #fc samp_rate/Decim,
         #bw "Filterwork Plot",
         #name
                1 #number of inputs
        ) self.qtgui_waterfall_sink_x_0.set_update_time(0.10)
        self._qtgui_waterfall_sink_x_0_win =
        sip.wrapinstance(self.qtgui_waterfall_sink_x_0.pyqwidget(), Qt.QWidget)
        self.top_layout.addWidget(self._qtgui_waterfall_sink_x_0_win)
        self.qtgui_sink_x_1 = qtgui.sink_f(
         1024, #fftsize
         firdes.WIN_BLACKMAN_hARRIS, #wintype
         0, #fc samp_rate/Decim, #bw
         "Quadrature Demod Plot", #name
         True, #plotfreq
         True, #plotwaterfall
         True, #plottime
         True, #plotconst
        ) self.qtgui_sink_x_1.set_update_time(1.0/10)
        self._qtgui_sink_x_1_win =
        sip.wrapinstance(self.qtgui_sink_x_1.pyqwidget(), Qt.QWidget)
        self.Display_grid_layout_2.addWidget(self._qtgui_sink_x_1_win, 0,1)
```

```
        self.qtgui_sink_x_0_0 = qtgui.sink_c(
         1024, #fftsize
         firdes.WIN_BLACKMAN_hARRIS, #wintype
         0, #fc
         samp_rate, #bw
         "Doppler-corrected Plot", #name
         True, #plotfreq
         True, #plotwaterfall
         False, #plottime
         False, #plotconst
        ) self.qtgui_sink_x_0_0.set_update_time(1.0/10)
        self._qtgui_sink_x_0_0_win =
        sip.wrapinstance(self.qtgui_sink_x_0_0.pyqwidget(), Qt.QWidget)
        self.Display_grid_layout_0.addWidget
        (self._qtgui_sink_x_0_0_win, 0,1)


        self.qtgui_sink_x_0 = qtgui.sink_c(
         1024, #fftsize
         firdes.WIN_BLACKMAN_hARRIS, #wintype
         0, #fc
         samp_rate, #bw
         "RX BaseBand Plot", #name
         True, #plotfreq
         True, #plotwaterfall
         False, #plottime
         False, #plotconst
        ) self.qtgui_sink_x_0.set_update_time(1.0/10)
        self._qtgui_sink_x_0_win =
        sip.wrapinstance(self.qtgui_sink_x_0.pyqwidget(), Qt.QWidget)
        self.Display_grid_layout_0.addWidget(self._qtgui_sink_x_0_win, 0,0)


        self.low_pass_filter_0 =
        filter.fir_filter_ccf(1, firdes.low_pass(
         1, samp_rate/Decim, 4900, 1000, firdes.WIN_HAMMING, 6.76))
        self.freq_xlating_fir_filter_xxx_3 =
        filter.freq_xlating_fir_filter_ccc(1, (1, ), shift, samp_rate)
        self.blocks_throttle_0 = blocks.throttle
        (gr.sizeof_gr_complex*1, samp_rate,True)
        self.blocks_multiply_xx_0 = blocks.multiply_vcc(1)
        self.blocks_file_source_0 = blocks.file_source
        (gr.sizeof_gr_complex*1,
        "/home/rsl/BeaconStation3/Captures/testooreos-beacon-master", True)
        self.blocks_file_sink_0 =
        blocks.file_sink(gr.sizeof_char*1, "/home/rsl/BeaconStation3/output.txt", False)
        self.blocks_file_sink_0.set_unbuffered(False)
        self.band_pass_filter_1 =
         filter.fir_filter_ccc(Decim, firdes.complex_band_pass(
         4, samp_rate,
FrequencyOffset*500,
FrequencyOffset*500+10000, 300, firdes.WIN_HAMMING, 6.76))
        self.audio_sink_0 = audio.sink(24000, "", True)
        self.analog_sig_source_x_0 =
        analog.sig_source_c
        (samp_rate/Decim, analog.GR_COS_WAVE, -1000*FrequencyOffset, 1, 0)
        self.analog_quadrature_demod_cf_0_0 =
```

```
        analog.quadrature_demod_cf(3.05577)
        self.analog_pwr_squelch_xx_0_0_0 =
        analog.pwr_squelch_cc(-100, 5e-1, 0, False)
        self.afsk_afsk1200_0 = afsk.afsk1200(samp_rate/Decim,4)

        ##################################################
        # Connections
        ##################################################
        self.connect
        ((self.blocks_multiply_xx_0, 0), (self.low_pass_filter_0, 0))
        self.connect
        ((self.afsk_afsk1200_0, 0), (self.blocks_file_sink_0, 0))
        self.connect
        ((self.low_pass_filter_0, 0), (self.analog_quadrature_demod_cf_0_0, 0))
        self.connect
        ((self.afsk_afsk1200_0, 0), (self.show_text_0_0_0_0, 0))
        self.connect
        ((self.low_pass_filter_0, 0), (self.qtgui_waterfall_sink_x_0, 0))
        self.connect
        ((self.analog_sig_source_x_0, 0), (self.blocks_multiply_xx_0, 1))
        self.connect
        ((self.blocks_file_source_0, 0), (self.blocks_throttle_0, 0))
        self.connect
        ((self.blocks_throttle_0, 0), (self.qtgui_sink_x_0, 0))
        self.connect
        ((self.freq_xlating_fir_filter_xxx_3, 0), (self.band_pass_filter_1, 0))
        self.connect
        ((self.band_pass_filter_1, 0), (self.blocks_multiply_xx_0, 0))
        self.connect
        ((self.freq_xlating_fir_filter_xxx_3, 0), (self.qtgui_sink_x_0_0, 0))
        self.connect
        ((self.analog_quadrature_demod_cf_0_0, 0), (self.audio_sink_0, 0))
        self.connect
        ((self.analog_quadrature_demod_cf_0_0, 0), (self.afsk_afsk1200_0, 0))
        self.connect
        ((self.analog_quadrature_demod_cf_0_0, 0), (self.qtgui_sink_x_1, 0))
        self.connect
        ((self.analog_pwr_squelch_xx_0_0_0, 0), (self.freq_xlating_fir_filter_xxx_3, 0))
        self.connect
        ((self.blocks_throttle_0, 0), (self.analog_pwr_squelch_xx_0_0_0, 0))


# QT sink close method reimplementation
    def closeEvent(self, event):
        self.settings = Qt.QSettings("GNU Radio", "SCU_RSL_BEACONv3")
        self.settings.setValue("geometry", self.saveGeometry())
        event.accept()

    def get_shift(self):
        return self.shift

    def set_shift(self, shift): self.shift = shift
        self.set_variable_qtgui_entry_0(self.shift)
        self.set_variable_qtgui_entry_1(self.frequency+self.shift)
        self.freq_xlating_fir_filter_xxx_3.set_center_freq(self.shift)

    def get_frequency(self):
        return self.frequency
```

```
def set_frequency(self, frequency): self.frequency = frequency
    self.set_variable_qtgui_entry_1(self.frequency+self.shift)

def get_variable_qtgui_entry_1(self):
    return self.variable_qtgui_entry_1

def set_variable_qtgui_entry_1(self, variable_qtgui_entry_1):
    self.variable_qtgui_entry_1 = variable_qtgui_entry_1
    Qt.QMetaObject.invokeMethod
    (self._variable_qtgui_entry_1_line_edit,
     "setText", Qt.Q_ARG("QString", str(self.variable_qtgui_entry_1)))

def get_variable_qtgui_entry_0(self):
    return self.variable_qtgui_entry_0

def set_variable_qtgui_entry_0(self, variable_qtgui_entry_0):
    self.variable_qtgui_entry_0 = variable_qtgui_entry_0
    Qt.QMetaObject.invokeMethod
    (self._variable_qtgui_entry_0_line_edit, "setText", Qt.Q_ARG("QString",
    eng_notation.num_to_str(self.variable_qtgui_entry_0)))

def get_samp_rate(self):
    return self.samp_rate

def set_samp_rate(self, samp_rate):
    self.samp_rate = samp_rate
    self.qtgui_sink_x_0_0.set_frequency_range
    (0, self.samp_rate)
    self.low_pass_filter_0.set_taps(firdes.low_pass
    (1, self.samp_rate/self.Decim, 4900, 1000, firdes.WIN_HAMMING, 6.76))
    self.qtgui_waterfall_sink_x_0.set_frequency_range(0, self.samp_rate/self.Decim)
    self.band_pass_filter_1.set_taps(firdes.complex_band_pass
    (4, self.samp_rate, self.FrequencyOffset*500, self.FrequencyOffset*500+10000, 300,
    firdes.WIN_HAMMING, 6.76))
    self.analog_sig_source_x_0.set_sampling_freq(self.samp_rate/self.Decim)
    self.qtgui_sink_x_1.set_frequency_range(0, self.samp_rate/self.Decim)
    self.qtgui_sink_x_0.set_frequency_range(0, self.samp_rate)
    self.blocks_throttle_0.set_sample_rate(self.samp_rate)

def get_FrequencyOffset(self):
    return self.FrequencyOffset

def set_FrequencyOffset(self, FrequencyOffset):
    self.FrequencyOffset = FrequencyOffset
    self.band_pass_filter_1.set_taps
    (firdes.complex_band_pass(4, self.samp_rate, self.FrequencyOffset*500,
    self.FrequencyOffset*500+10000, 300, firdes.WIN_HAMMING, 6.76))
    self.analog_sig_source_x_0.set_frequency
    (-1000*self.FrequencyOffset)
    Qt.QMetaObject.invokeMethod
    (self._FrequencyOffset_counter,
     "setValue", Qt.Q_ARG("double", self.FrequencyOffset))
    Qt.QMetaObject.invokeMethod
    (self._FrequencyOffset_slider, "setValue",
    Qt.Q_ARG("double", self.FrequencyOffset))

def get_Decim(self):
```

```
        return self.Decim

    def set_Decim(self, Decim):
        self.Decim = Decim
        self.low_pass_filter_0.set_taps
        (firdes.low_pass
        (1, self.samp_rate/self.Decim, 4900, 1000, firdes.WIN_HAMMING, 6.76))
        self.qtgui_waterfall_sink_x_0.set_frequency_range(0, self.samp_rate/self.Decim)
        self.analog_sig_source_x_0.set_sampling_freq(self.samp_rate/self.Decim)
        self.qtgui_sink_x_1.set_frequency_range(0, self.samp_rate/self.Decim)

if __name__ == '__main__':
    import ctypes
    import sys
    if sys.platform.startswith('linux'):
        try:
            x11 = ctypes.cdll.LoadLibrary('libX11.so')
            x11.XInitThreads()
        except:
            print "Warning: failed to XInitThreads()"
    parser = OptionParser(option_class=eng_option, usage="%prog: [options]")
    (options, args) = parser.parse_args()
    qapp = Qt.QApplication(sys.argv)
    tb = SCU_RSL_BEACONv3()
    tb.start()
    tb.show()
    def quitting():
        tb.stop()
        tb.wait()
    qapp.connect(qapp, Qt.SIGNAL("aboutToQuit()"), quitting)
    qapp.exec_()
    tb = None #to clean up Qt widgets
```